

THINK's LightspeedCTM

The Professional's Choice

S T A N D A R D
L I B R A R I E S
R E F E R E N C E

For Macintosh Plus,
Macintosh SE, and
Macintosh II computers

THINK'S LightspeedC

The Professional's Choice

STANDARD
LIBRARIES
REFERENCE

Credits

Software: Michael Kahl

User's Manual: Philip Borenstein

Product Manager: Diana Bury

Special thanks to Andrew Singer.

Copyright © 1988 Symantec Corporation. All Rights Reserved
THINK Technologies Division
135 South Road
Bedford, MA 01730, USA
(617) 275-4800

The product names mentioned in this manual are the trademarks or registered trademarks of their manufacturers.

"Lightspeed" is a registered trademark of Lightspeed, Inc., and is used with its permission.

ResEdit, RMaker, and Macsbug are copyrighted programs of Apple Computer, Inc. licensed to Symantec Corp. to distribute for use only in combination with THINK's LightspeedC. Apple software shall not be copied onto another diskette (except for archive purposes) or into memory unless as part of execution of THINK's LightspeedC. When THINK's LightspeedC has completed execution, Apple Software shall not be used by any other program.

APPLE COMPUTER, INC. MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED REGARDING THE ENCLOSED SOFTWARE PACKAGE, ITS MERCHANTABILITY OR ITS FITNESS FOR ANY PARTICULAR PURPOSE. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED IN SOME STATES. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY PROVIDES YOU WITH SPECIFIC LEGAL RIGHTS. THERE MAY BE OTHER RIGHTS THAT YOU MAY HAVE THAT VARY FROM STATE TO STATE.

Contents

1	Standard Libraries	1
2	stdio	3
3	strings	73
4	math	99
5	storage	127
6	unix	135
7	unix_strings	187
8	storageu	197
	index	211
	license agreement	213

Standard Libraries

1

Introduction

This manual describes the standard library functions. These functions are included to provide compatibility and portability with other computer systems. The term “standard” here is something of a misnomer since there is no universally accepted set of library routines. The functions included here are a subset of those found in most popular C environments.

Aside from porting code from other machines, you might find the standard libraries useful for “quick-and-dirty” programs where you don’t really want to go through the Macintosh interface.

The Libraries

The libraries are supplied in project form so the smart linker can create the smallest code possible. The source code for all the standard libraries is included in your THINK C package, so you can modify the routine to suit your programs.

The libraries are:

stdio	Miscellaneous standard I/O functions.
strings	String handling functions.
math	Math functions.
storage	Memory allocation functions.
unix	Miscellaneous functions provided for UNIX compatibility.
unix_strings	UNIX string handling functions.
storageu	UNIX memory allocation functions.

Note: If you are using the 68881 code generation, be sure you use the xxx881 versions of the standard libraries.

To use the functions in these libraries, you need add the libraries to your project and include the appropriate header file in your program. The appropriate `#include` statement is shown as a part of the syntax for each function.

A function index in the back of the manual that will help you to quickly locate a function you need.

Standard Libraries Reference

stdio 2

Introduction

The `stdio` library provides the standard I/O routines found in most C environments. Functions that normally send their output to the console or to `stdout` open a window called console in THINK's LightspeedC.

Note: Since the `stdio` use the Macintosh routines to create the console window and to write into it, be sure you add MacTraps to your project.

Defined Symbols

The following symbols are defined by including `stdio.h`:

BUFSIZ

`BUFSIZ`, the size of an input/output buffer, is defined as 512. Buffers are used in calls to functions like `read()` and `write()`.

Buffers less than `BUFSIZ` are less efficient, buffers larger than `BUFSIZ` are treated as sets of `BUFSIZ` blocks. `BUFSIZ` is measured in characters (or bytes), so a buffer will hold proportionately fewer items of a larger data type.

`BUFSIZE` is an alternate symbol with the same definition.

_console

`_console` contains the file pointers for I/O to the console. The console is the keyboard and the screen. `stdin`, `stdout`, and `stderr` initially point to the console as well, but can be redirected to point to a file.

A family of functions beginning with the letter `c` (`cgets()`, `cprintf()`, etc.) are dedicated to performing I/O to the console.

EOF

`EOF`, the value used to represent the end of a file, is defined as `-1`. Write code using `EOF` rather than `-1`. This will allow code to be ported to a system which does not use `-1` to represent the end of a file (some systems use `0` to represent `EOF`). `EOF` is returned by the appropriate library functions when they are at the end of a file.

For example:

Standard Libraries Reference

```
while ((ch = getchar()) != EOF)
{
}
```

errno

errno is a global variable, located in stdio, holding the last error message number from any stdio I/O routine.

Note: See the following "Error Messages" section.

FILE

FILE, the structure which holds the file descriptor, is defined below. This type definition creates a type called FILE which can be used anywhere you could use one of C's built in types (int, char, long, ...). Because of the library functions, you will probably never have to access the internals of a FILE directly. The fields are included here just in case.

Note: This manual will use "file descriptor" to refer to the C structure which holds the file information. Kernighan and Ritchie use "file descriptor" to refer to the number assigned to each open file. This manual will call that number the "file number". A "file number" is a pointer to a file descriptor (FILE *).

```
typedef struct {
    int refnum; /* Operating System (OS) ref number */
    int last_error; /* Last error produced by access to this file */
    unsigned user_sys:1; /* Single bit flags indicating: */
        /* user buffer instead of system buffer */
    InUse:1; /* this file descriptor is in use */
    StdStream:1; /* standard stream (keyboard, CRT,...) */
    rd:1; /* read permission */
    wr:1; /* write permission */
    look_full:1; /* look ahead character is valid */
    char look_ahead; /* Next char to be returned by getc() */
    Handle filebuf; /* Handle of OS buffer for this file */
} FILE;
```

_file

_file is an array of FILE (_file[_NFILE]) which is used to hold the file descriptors of all the files which are open. _file is used implicitly by all the file access functions. When you use open() to open a file, a FILE in _file will be dedicated to that file, and the index will be returned as a file number. If you use fopen() to open a file, the file pointer (FILE *) to the FILE in _file is returned instead of the file number. File I/O functions require either a file number or a file pointer as an argument. fileno() returns the corresponding file when

given a file pointer. To go the other way, use `_file[fn]`, the file pointer corresponding to file number `fn`.

See also: `FILE`, `fopen()`, `open()`, `fileno()`

`_NFILE`

`_NFILE`, the number of files that can be open at one time, is defined as 15.

`NULL`

`NULL`, the value of a pointer that is pointing to nothing, is defined as 0L. `NULL` is returned by many library functions (eg: `calloc()` or `sbrk()`). It is typically used to indicate an error by functions which return pointers. On every system, `NULL` will point to nothing, although its value will not always be 0L.

`stdin`, `stdout`, `stderr`

The three standard I/O file pointers (to the input, output, and error) may be used anywhere that a function takes a `FILE *` as an argument. For example, `getc(stdin)` gets a single character from `stdin`; `fputs("Hello", stdout)` puts the string `Hello` on the standard output. Some functions use the standard I/O automatically, most notably: `getchar()` and `putchar()`, `gets()` and `puts()`, `scanf()` and `printf()`. Library functions send their error messages to `stderr` automatically; your procedures can use it too or can send error messages into other files. The tables of input and output functions in the introduction to this section show which functions to use for `stdin` and `stdout`.

I/O from/to `stdin` and `stdout` can be redirected so that a file is used instead of the console. This allows programs to be written and compiled using `stdin` and `stdout`, and then connected to a file at run time.

Experienced C programmers from other systems should take note of the `cxxxxx` family of functions (`cputs`, `cprint`, `cgets`, `cscanf`, as well as `putch`, `getch`, `getche`) which send their output to the console automatically. I/O to/from the console using these functions cannot be redirected. The console I/O functions are included in the tables in the introduction.

- GET A STRING FROM THE CONSOLE

cgets

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
char *cgets(char *s);
```

DESCRIPTION

`cgets()` gets a string from the console (the keyboard) and puts it into the character string pointed to by `s`. The string is terminated when a carriage return (`0x0D`, `'\r'`) is entered from the keyboard. The input is always echoed to the screen.

(Note that this is not the usual C definition of a string. The carriage return is stripped off by `cgets`, and the string is stored internally in the standard C fashion - as a character array terminated by a null character (`'\0'`)).

Be sure that `s` is a character array long enough to handle any anticipated input, or else the excess input will be written into the memory that follows `s`, destroying whatever was there before.

`cgets()` echoes input to the screen.

RETURN VALUE

A pointer to string `s`; `NULL` if error.

SEE ALSO

`cputs()`

- CLEAR LAST ERROR FLAG OF A STREAM

clearerr

clrerr

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
void clearerr(FILE *stream);

void clrerr(FILE *stream);
```

DESCRIPTION

`clearerr()` resets the `last_error` flag of the file descriptor of the given stream. The `last_error` flag could be accessed directly by the syntax:

```
stream -> last_error
```

`stream` can be any pointer to a `FILE`.

`clrerr()` is a synonym for `clearerr()`. It too clears the last error flag of a given stream.

SEE ALSO

`FILE`

- ENABLE/DISABLE "CLICK TO CONTINUE" ON EXIT

Click_on

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
void Click_On(Boolean flag);
```

DESCRIPTION

This function controls the "click mouse to continue" option. If flag is 1, the option is enabled; if flag is 0 the option is disabled. If this option is enabled, when a program that uses stdio terminates, a window (titled "Exit Window") will appear. Only when the user clicks in the close box, presses the return key, or (if the menus were created by stdio) selects the Quit command from the File menu will the program actually exit. This option is initially enabled.

- CLOSE ALL OPEN FILES

_closeall

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int closeall(void);
```

DESCRIPTION

`_closeall ()` closes all open files that were opened via a call to `open ()` or `fopen ()`.

`_closeall ()` is called automatically upon successful program termination.

Call `close ()` or `fclose ()` to close a single file.

RETURN VALUE

The number of files that could not be closed.

SEE ALSO

`close ()`, `fclose()`, `open()`, `fopen()`

- FORMATTED PRINT TO THE CONSOLE SCREEN

cprintf

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int  cprintf(char *format, ...);
```

DESCRIPTION

`cprintf()` does a `printf` to the console screen. `printf` stands for formatted print, and is the most general way to output text and variables, whether to the console (`cprintf()`), a file (`fprintf()`), `stdout` (`printf()`) or to a string in memory (`sprintf()`). See `printf()` for a complete description of the format rules.

RETURN VALUE

The number of characters sent to the output stream if successful; EOF if an error occurs.

SEE ALSO

`printf()`, `sprintf()`, `fprintf()`

- PUT STRING TO THE CONSOLE SCREEN

cputs

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
void cputs(char *s);
```

DESCRIPTION

`cputs()` puts a null terminated ('\0') string on the user's console screen. Unlike `puts()`, `cputs` does NOT automatically send a carriage return ('\r') or a newline ('\n') when it reaches the end of the string. However, the string may explicitly include '\r' or '\n' characters. The string can be a variable or a constant, as shown in the example. The null character is not written to the console.

EXAMPLES

```
#include <stdio.h>
char *s, buf[20]
s = gets(buf);
cputs(s);
cputs("\r\nDONE\r\n");
```

SEE ALSO

`cgets()`, `puts()`, `fputs()`

- READ TEXT AND DATA FROM CONSOLE

cscanf

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int  cscanf(char *format, ...);
```

DESCRIPTION

`cscanf()` does a `scanf()` from the keyboard of the console. `scanf()` stands for formatted scan, and is the most flexible way to read text in and convert it into a mixture of strings and numeric values.

Note that `cscanf()` takes a variable number of arguments -- it can read values into a variable number of variables. `format` is a string telling what sort of variables to expect, in what order, and with what text intermixed.

NOTE: `args` are POINTERS to the variables to read into, NOT the variables themselves. Remember to use the `&` operator to get the address of a variable. If you don't, you will overwrite random memory.

Refer to the description of `scanf()` for complete information about the `format` and `args` arguments.

`scanf()` and `printf()` are conceptually inverse functions (and therefore, `cscanf()` is to `scanf()` as `cprintf()` is to `printf()`).

RETURN VALUE

Number of items successfully input.

SEE ALSO

`scanf()`, `fscanf()`, `sscanf()`, `printf()`

- TESTS WHETHER CHARACTER C HAS CHARACTERISTIC TYPE

ctype

LIBRARY

stdio

```
#include <stdio.h>
int istype(char c);
```

DESCRIPTION

There is a family of functions to make various tests on a character. All take a single character as an argument and return an `int` value. The return value is non-zero if the condition is true, 0 if it is false. The `istype` functions (with all numeric values in decimal) are:

<code>isalnum(c)</code>	True if <code>c</code> is a letter (a-z, A-Z) or a digit (0-9)
<code>isalpha(c)</code>	True if <code>c</code> is a letter (a-z, A-Z)
<code>isascii(c)</code>	True if <code>c</code> is an ASCII character (ASCII 32-128)
<code>isctrl(c)</code>	True if <code>c</code> is a control character (ASCII 0-31)
<code>iscsym(c)</code>	True if <code>c</code> is a character that can appear in a valid C identifier (a letter, digit, or underscore)
<code>iscsymf(c)</code>	True if <code>c</code> is a character that can appear as the first character of a valid C identifier (a letter or an underscore)
<code>isdigit(c)</code>	True if <code>c</code> is a digit (0-9)
<code>isgraph(c)</code>	True if <code>c</code> is any printing character other than a space (ASCII 041 through 0176)
<code>islower(c)</code>	True if <code>c</code> is a lower case letter (a-z)
<code>isodigit(c)</code>	True if <code>c</code> is an octal digit (0-7)
<code>isprint(c)</code>	True if <code>c</code> is a printable character (ASCII 32-255; but not 127)
<code>ispunct(c)</code>	True if <code>c</code> is a punctuation character. A punctuation character is one of: # \$ % & * () " ' \ ` ~ : ; , < > / ? \ [] { } - _ = + .
<code>isspace(c)</code>	True if <code>c</code> is a space character. A space character is one of: space, \t, \v, \n, \f or \r.
<code>isupper(c)</code>	True if <code>c</code> is an upper case letter (A-Z)
<code>isxdigit(c)</code>	True if <code>c</code> is a hex digit (0-9, A-F, a-f).

Standard Libraries Reference

The functions all use a predefined table called `__ctype` which contains a mask bit for each type of character, as listed below:

<code>_alpha_</code>	1	<code>_ascii_</code>	16
<code>_digit_</code>	2	<code>_cntrl_</code>	32
<code>_hex_</code>	4	<code>_punct_</code>	64
<code>_octal_</code>	8	<code>_space_</code>	178

With the exception of `isupper()` and `islower()`, which are true functions, these routines are macros that test the mask settings for the respective character.

RETURN VALUE

non-zero if the condition is true; 0 if the condition is false.

SEE ALSO

`toascii()`, `toupper()`, `tolower()`

- TURN PRINTER ECHO ON OR OFF

Echo_to_Printer

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
void Echo_to_Printer(Boolean flag);
```

DESCRIPTION

Call this function with the argument TRUE to turn on printer echoing. When printer echoing is on, the console window text output will be echoed to the printer. If input is being echoed to the console window, it will be echoed to the printer as well.

Call this function with the argument FALSE to turn off printer echoing. This is the default condition.

- CLOSE A FILE

fclose

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int fclose(FILE *stream);
```

DESCRIPTION

`fclose()` closes the given `stream`, flushing the I/O buffer if necessary. The file descriptor in `_file` used by `stream` is then available for another file.

The difference between `fclose()` and `close()` is that `fclose()` takes a file pointer as its argument while `close()` takes the file descriptor number.

RETURN VALUE

0 if success; EOF if error.

SEE ALSO

`open()`, `close()`, `_closeall()`

- TEST WHETHER AT END OF STREAM

feof

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int feof(FILE *stream);
```

DESCRIPTION

`feof()` is used to test whether a previous I/O call to the given `stream` caused an EOF error (such as trying to get a character after having reached the end of the file).

Naturally, if you use `clrerr()` or `clearerr()` to clear the error flag of that `stream`, `feof()` will no longer report an EOF error. Similarly, a later call with another type of error will also erase the EOF error flag.

RETURN VALUE

non-zero if `last_error` flag in the `stream` structure is an EOF error;
0 if `last_error` is not an EOF error.

SEE ALSO

`FILE`, `clrerr()`, `clearerr()`, `ferror()`

- TEST WHETHER I/O ERROR FLAG IS SET FOR STREAM

ferror

LIBRARY	stdio
SYNTAX	<pre>#include <stdio.h> int ferror(FILE *stream);</pre>
DESCRIPTION	<code>ferror()</code> tests whether any sort of I/O error flag is set for the given stream. <code>clrerr()</code> and <code>clearerr()</code> will reset the error flag.
RETURN VALUE	non-zero if any I/O error had occurred; 0 if the <code>last_error</code> flag in the stream structure is not set.
SEE ALSO	<code>feof()</code> , <code>clrerr()</code> , <code>clearerr()</code>

- FLUSH I/O BUFFER OF STREAM

fflush

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int fflush(FILE *stream);
```

DESCRIPTION

`fflush()` flushes the I/O buffer of the specified stream. This is often done if an I/O error has occurred and you want to start anew. Buffers should also be flushed before closing files.

Note that `fflush()` returns 0 if successful.

RETURN VALUE

0 if success; EOF if an error occurred while trying to flush the buffer.

SEE ALSO

`fopen()`

- GET A CHARACTER FROM STREAM

fgetc

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int fgetc(FILE *stream);
```

DESCRIPTION

`fgetc()` reads in the next character from the given `stream`. Since `fgetc()` returns the integer value of the character, it can be used to get bytes from a binary file.

Use `fread()` or `fgets()` to read in multiple characters more easily and efficiently, and use `fscanf()` to read text and data directly into variables.

RETURN VALUE

EOF if error or at the EOF of the stream; the integer value of the character otherwise.

SEE ALSO

`fread()`, `fgets()`, `fscanf()`, `getc()`, `gets()`, `read()`, `scanf()`

- GET A STRING FROM STREAM AND PUT IN ARRAY

fgets

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
char *fgets(char *s, int n, FILE *stream);
```

DESCRIPTION

`fgets()` tries to read the next `n` characters from `stream` and store them in the character array `s`. It adds the null character (`'\0'`) to the character array after the `n`th character, and returns a pointer to `s`. Therefore, `s` must be declared as a character array large enough to hold `n+1` characters.

Two events will stop `fgets()` before it has read in `n` characters. First, if it reads in a newline (`'\n'`), it adds the `'\n'` to the string, adds the `'\0'`, and returns `s`. Second, if `fgets()` reads in an EOF, it considers it an error, sets the EOF error flag, stops trying to read in characters from the file, and returns a NULL pointer.

Use `ferror` to find out which error occurred last.

RETURN VALUE

NULL if error or if EOF; `s` (the first argument) otherwise.

SEE ALSO

`fgetc()`, `fread()`, `fscanf()`, `ferror()`

- OPEN A FILE

fopen

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
FILE *fopen(char *pathname, char *type);
```

DESCRIPTION

`fopen()` opens a stream to `pathname` by creating a file descriptor as well as doing the necessary operating system work. `fopen()` returns the pointer to the newly created file descriptor. The file is opened according to the `type` (the values for `type` are described below). The effect of `type` depends on whether the given `pathname` can be found: a new file is created if `pathname` does not exist, `pathname` is opened if it already exists.

Files can also be opened with `open()`, which returns the file number in the array of file descriptors rather than the pointer to the file descriptor. Use `open()`, `close()`, etc. instead of the standard C library file calls `fopen()`, `fclose()`, etc. when you are concerned with UNIX compatibility.

`fopen()` and `open()` are not synonymous. Not only do they interpret their arguments differently, their arguments are of different types. Note that some library functions access a file by its file pointer, others by its file number. Therefore, your choice of `fopen()` vs. `open()` will dictate your choice of other library routines.

The values for `type` are strings and are:

- "r" open an existing file for reading.
- "w" create a file for writing if it does not exist; delete and then recreate a file for writing if it already exists.
- "a" create a file for writing if it does not exist; open a file and place pointer at the EOF for appending if it already exists.
- "r+" open an existing file for reading and writing.
- "w+" create a file for reading and writing if it does not exist; delete and recreate a file for reading and writing if it already exists.

"a+" create a file for reading and writing if it does not exist; open a file and place pointer at the EOF for appending and reading if it already exists.

b binary file: '\r' not converted to '\n' on reads, '\n' not converted to '\r' on writes.

b is not in quotes because it is added to one of the other values for type. For example, "rb", "wb", "ab", "r+b", "w+b", "a+b". "b" alone is meaningless and is an error.

Note that the type argument is a character pointer, not a single character. Therefore, `newfile = fopen("foo", "r")` is legal while `newfile = fopen("foo", 'r')` will not operate as intended, and will have unpredictable results.

RETURN VALUE

NULL if error; the pointer to the file descriptor if success.

SEE ALSO

`open()`, `_file`, `FILE`, `fileno()`, `fclose()`, `fflush()`, `fread()`, `fwrite()`

OPEN A NEW STDIO WINDOW

fopenw

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
FILE *fopenw(char *title, Point upperLeftCorner,
StdWindowOptions *optionsPtr);
```

DESCRIPTION

fopenw opens a new window with the specified title at the designated position (in global coordinates).

optionsPtr is a pointer to a StdWindowOptions structure as defined in fopenw.h. If the pointer is NULL (0L), then the window will be created with the following defaults: a 24-by-80 window, visible cursor, input echoed, tab width of 4 spaces, window brought forward on output to it (including echoing of input), grow box, draggable, go-away box, scrolling, and line wrap-around. (On big-screen Macintoshes, the default screen will be larger.) After calling fopenw(), the memory pointed to by optionsPtr is no longer needed.

RETURN VALUE

Returns an ordinary FILE pointer variable which can be passed to fprintf, fclose, etc.

- FORMAT AND PRINT TO A FILE

fprintf

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int fprintf(FILE *stream, char *format, ...);
```

DESCRIPTION

`fprintf()` formats and prints output to the specified `stream`. It is used to print a combination of text and data to the file pointed to by `stream`. In all other ways, it is identical to `printf()` which does a formatted print to the standard output. See `printf()` for a description of the format and optional `args` arguments.

`fputc()`, `fputs()`, and `fwrite()` put a character, a string, and a block of text in a file, respectively: they are simpler and more efficient, but are not as general and cannot deal with numeric data.

`fscanf()` is the inverse of `fprintf()`: it does a formatted read of text and data from a file.

RETURN VALUE

The number of characters sent to the output `stream` if successful; EOF if an error occurs.

SEE ALSO

`printf()`, `cprintf()`, `sprintf()`, `fputc()`, `fputs()`, `fwrite()`, `fscanf()`

- PUT A CHARACTER INTO A FILE

fputc

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int fputc(char c, FILE *stream);
```

DESCRIPTION

`fputc()` adds a single character `c` to `stream`. See `fputs()`, `fwrite()`, and `fprintf()` for ways to add strings, blocks of text, or combinations of text and data to a file. `putc()` puts a single character to the standard input.

RETURN VALUE

EOF if error (such as a read only file); `c`, the integer value of the character, if success.

SEE ALSO

`putc()`, `fputs()`, `fwrite()`, `fprintf()`

- PUT A STRING INTO A FILE

fputs

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int fputs(char *s, FILE *stream);
```

DESCRIPTION

`fputs()` puts a null terminated (`'\0'`) string to `stream`. The `'\0'` is not written to the file.

RETURN VALUE

0 if successful; otherwise EOF.

SEE ALSO

`cputs()`, `puts()`, `fprintf()`, `fwrite()`, `fputc()`

- READ A BLOCK OF CHARACTERS FROM A FILE

fread

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int fread(char *ptr, unsigned size_of_ptr, int
count, FILE *stream);
```

DESCRIPTION

`fread()` tries to read `count` items from `stream` and store them in the block pointed to by `ptr`. The `size_of_ptr` field should be an unsigned integer that is the size of the object pointed to by the pointer. If `fread` reaches the end of the file before it has read `count` bytes, it sets the EOF error flag and returns the number of bytes it was able to read.

`read()` is a similar function to read a block of data from a file. `read()` accesses the file by its file number rather than its file pointer. `fscanf()`, `fgetc()`, `fgetc()` and `getc()` are other functions to get data from a file.

EXAMPLES

```
#include <stdio.h>
struct FOO {
    int a;
    char b;
} foo[3];
fread (&foo[0], sizeof(struct FOO), 3, stdin);
/* reads 3 items of sizeof FOO into buffer
   pointed to by foo from stdin */
```

RETURN VALUE

0 on immediate end of file or error (use `ferror()` and `feof()` to tell); otherwise the number of items transferred.

SEE ALSO

`read()`, `fscanf()`, `fgetc()`, `fgetc()`, `getc()`

- REDIRECT OUTPUT TO DIFFERENT FILE

freopen

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
FILE *freopen(char *filename, char *type, FILE
*stream);
```

DESCRIPTION

`freopen()` closes the file whose file descriptor is `stream` and calls `fopen()`, passing the `filename` and `type`. The file will be opened using the same string that was used to close it.

This routine is used to redirect a `stream`'s output to a different file. For example, after the call `freopen("abc", "w", stdout)`, all output to `stdout` will go to the file "abc".

RETURN VALUE

`NULL` if error (sets an error code in `errno`); otherwise `stream`.

SEE ALSO

`fopen()`, `fclose()`

- SCAN AND FORMAT INPUT FROM A FILE

fscanf

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int fscanf(FILE *stream, char *format, ...);
```

DESCRIPTION

`fscanf()` does a formatted scan taking input from `stream`. It breaks the input up into characters, strings, integers, etc. according to `format`, and stores that formatted data in the variables pointed to by `args`. There can be a variable number of `args`. The `args` must be pointers to variables, not the variables themselves. See `scanf()` for a description of the `format` argument.

CAUTION: The most common mistake when using `fscanf()` is to use the variables themselves rather than the pointers to the variables. `fscanf()` needs to know the address of the variables it will use to store data, not their present values. Failing to use addresses to store the values will overwrite random memory. See the examples in `scanf()`. If `fscanf()` runs out of arguments before reaching the end of `s`, arbitrary values from the stack are used as the address of the arguments and overwrite random memory.

Similar functions are `scanf()` (from `stdio`), `cscanf()` (from the console), and `sscanf()` (from a string). Other functions to get data from a file include `getc()`, `fgetc()`, `fgets()`, `read()`, and `fread()`. `fprintf()` does a formatted print of data to a file.

RETURN VALUE

Number of items successfully input.

SEE ALSO

`scanf()`, `cscanf()`, `sscanf()`

- MOVE TO A DIFFERENT POINT WITHIN A FILE

fseek

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int fseek(FILE *stream, long offset, int type);
```

DESCRIPTION

`fseek()` allows for non-sequential I/O to a stream. When a file is opened for reading or writing, a "logical cursor" is placed at the beginning of the file. That logical cursor is advanced through the file with each read or write. `fseek()` and its cousin `lseek()` move that logical cursor. (`lseek()` differs in two main ways: it takes a file number rather than a pointer to the file descriptor and it returns a long rather than an int).

The type argument determines where the seek begins:

- 0 offset relative to the beginning of the file
- 1 offset relative to current file position
- 2 offset relative to the end of file

`offset` is simply the count in bytes from the starting point specified by the type argument. Note that `offset` is a long, and can be either positive or negative.

EXAMPLES

```
FILE *fd;
fseek(fd, 0L, 0); /* move to start of file */
fseek(fd, 20L, 1); /* move forward in file 20 bytes */
fseek(fd, 0L, 2); /* move to end of file */
```

RETURN VALUE

Zero if success, non-zero if error.

SEE ALSO

`lseek()`

- RETURN CURRENT POSITION WITHIN FILE

ftell

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
long ftell(FILE *stream) ;
```

DESCRIPTION

`ftell ()` returns the current position within the file. `ftell ()` returns `-1L` if an error has occurred.

RETURN VALUE

Current position within file if successful; `-1L` if error.

SEE ALSO

`tell ()`, `fseek ()`

- WRITE A BLOCK OF CHARACTERS TO A FILE

fwrite

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int fwrite(char *ptr, unsigned size_of_ptr, int
count, FILE *stream);
```

DESCRIPTION

`fwrite()` writes a block of data to `stream`. `count` is the number of items to be written, and `ptr` is a pointer to the start of the block. `size_of_ptr` is the size of the item to write.

`write()` also writes a block of data to a file. `write()` differs in two ways: it accesses the file by its file number and it simply writes a specified number of bytes to the file.

`fprintf()`, `fputs()`, `fputc()`, and `putc()` are other functions which add data to a file. `fread()` is the inverse of `fwrite()`.

RETURN VALUE

0 if error (check with `ferror()` and `feof()`); otherwise number of items transferred.

SEE ALSO

`fread()`, `write()`, `fprintf()`, `fputs()`, `fputc()`, `putc()`

- GET NEXT CHARACTER FROM FILE

getc

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int getc(FILE *stream) ;
```

This is a macro calling `fgetc()`

DESCRIPTION

`getc()` gets the next character from `stream`. `getc()` returns that character as an `int` so that it can be used to get bytes from a binary file.

`fgetc()` also gets a character from a file, while `fscanf()`, `fread()`, `fgets()` and `read()` get larger groups of data. `putc()` is the inverse of `getc()` and adds a character to a file.

RETURN VALUE

The integer value of the next character from `stream`; EOF on end of file or error.

SEE ALSO

`fgetc()`, `fgets()`, `fscanf()`, `fread()`, `read()`, `putc()`

- GET NEXT CHARACTER FROM KEYBOARD, DON'T ECHO

getch

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int getch(void);
```

DESCRIPTION

`getch()` gets the next character from the keyboard. The character is not echoed. (`getche()` is the same as `getch()` except the character is always echoed to the screen).

Note that the return value is an `int`, not a `char`. This is so that `getch()` can return all possible characters (0-255) and EOF (-1).

RETURN VALUE

The integer value of the character; or EOF on end of file or error.

SEE ALSO

`getchar()`, `getche()`

- GET NEXT CHARACTER FROM STDIN

getchar

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int  getchar()
```

`getchar()` is a macro calling `fgetc(stdin)`

DESCRIPTION

`getchar()` gets the next character from `stdin`. The character is not echoed to `stdout`. Note that the return value is an `int`, not a `char`. This is so that `getchar()` can return all possible characters (0-255) and EOF (-1).

RETURN VALUE

The integer value of the character; or EOF on end of file or on error.

SEE ALSO

`getc()`, `getch()`, `getchar()`, `getche()`, `fgetc()`

- GET NEXT CHARACTER FROM KEYBOARD, ECHO TO SCREEN

getche

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int getche(void);
```

DESCRIPTION

`getche()` is the same as `getch()` except that it echoes the character to the screen. See the description of `getch()`.

RETURN VALUE

The integer value of the character from the keyboard; or EOF on end of file.

SEE ALSO

`getc()`, `getch()`, `getchar()`, `getche()`, `fgetc()`

- GET CHARACTERS FROM STDIN TILL NEWLINE AND STORE IN S

gets

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
char *gets(char *s)
char *s;
```

DESCRIPTION

`gets()` reads characters in from `stdin` to the character array `s`. When it reads in a newline (`'\n'`), the `'\n'` is discarded and a null character (`'\0'`) is added to `s` to terminate the string. Therefore, `s` must be large enough to hold one more character than is anticipated.

`fgets()` and `cgets()` get strings from a file and from the console respectively. They have some differences so check their descriptions.

`getchar()` and `scanf()` also read in data from `stdin`.

RETURN VALUE

`s`, the first argument, is returned if success; `NULL` if EOF or error (use `ferror()` and `feof()` to determine cause).

SEE ALSO

`cgets()`, `fgets()`, `getchar()`, `scanf()`

- GET CURSOR POSITION

getxpos

getypos

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int getxpos(void)
int getypos(void)
```

DESCRIPTION

These functions return the x and y coordinates of the cursor in the current window.

RETURN VALUE

x or y coordinates of the cursor.

SEE ALSO

setwindow

- GET THE SCREEN BUFFER OF STDIO WINDOW

Get_ScreenPtr

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
char *Get_ScreenPtr(FILE *windowFileVar);
```

DESCRIPTION

This function returns the address of the screen buffer for the window associated with the FILE variable windowFileVar. The buffer can be treated as a two-dimensional array of characters whose size is given by the maxcol and maxrow fields of the StdWindowOptions structure defined in fopenw.h.

RETURN VALUE

The address of the screen buffer associated with windowFileVar.

- MOVE CURSOR ON THE SCREEN

gotoxy

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
void gotoxy(int x,int y);
```

DESCRIPTION

`gotoxy()` simply moves the cursor on the screen to the given (x,y) character position (as in standard I/O).

0,0 is in the top left corner of the screen. The range of possible coordinates will depend on the point size. For the default font (25 lines by 80 characters), the bottom right corner coordinates will be 80,25. Character positions are calculated for a monospaced font.

Note that this is different from `move()`, which uses a pixel-based coordinate system.

SEE ALSO

`circle()`, `label()`, `line()`, `move()`, `point()`

- TEST WHETHER KEYBOARD WAS HIT

kbhit

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int kbhit(void);
```

DESCRIPTION

`kbhit()` tests whether there is a character available from the keyboard without getting the character. `kbhit()` is provided so that you can write a loop to watch until the keyboard is hit, doing other work or with a timeout. `getch()` and similar functions have no timeout, and so take control for an indefinite period of time. Once the keyboard is hit, you can use `getch()` and know that it will return.

RETURN VALUE

1 if a keyboard character is available; 0 if no keyboard character is available.

SEE ALSO

`getch()`, `getche()`

- CALL A PROCEDURE BEFORE PROGRAM EXIT

onexit

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
typedef void (*Proc) ();
Proc *onexit (Proc proc)
```

DESCRIPTION

Arrange for C function `Proc` to be called just before program exit. `Proc` is called with no arguments. Up to 32 `Proc` routines may be specified. They are called in the reverse order in which they were supplied to `onexit()`.

RETURN VALUE

`NULL` if `onexit()` was unsuccessful; or `Proc` if successful.

SEE ALSO

`exit()`, `_exit()`

• PRINT AND FORMAT TO STDOUT

printf

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int printf(format [, args])
char *format;
```

DESCRIPTION

`printf()` is the easiest way to output a mixture of text and values of variables (character, numeric, and string) to `stdio`. Other versions of `printf()`, such as `fprintf()`, `sprintf()`, `cprintf()`, `vprintf()`, `vfprintf()`, `vsprintf()`, and `vcprintf()`, use the same argument conventions except as noted in their own descriptions. The versions differ in where the output is sent.

`printf()` takes a variable number of arguments. The first argument, `format`, is required. It is a string containing text and format specifiers. The format specifiers tell how to interpret and format the variable number of `args` that follow.

The `args` argument(s), written in braces above to indicate that it is optional and may be repeated, is a list of all the variables that this `printf()` statement will take data from. Make sure that there is one variable argument of the correct size for each format specifier in `format` requiring data. Arguments must also be of the right type. For example, if the format specifier specifies a string, the argument must be a `char*`. If the number of arguments, or the argument types, do not match the number and type of format specifiers, expect unpredictable results.

Format specifiers begin with the character `%` and include zero or more of the following conversion specification elements:

```
% [option flags] [field size] [.precision] [size]
conversion
```

Some of these elements are optional, but if present, they must be specified in the order in which they are described below.

Option Flags (optional):

- Left adjust output in field, pad on right (default is to right justify).

- 0 Use zero (0) rather than space for the pad character.
- + Always produce a sign, either + or —
- space Always produce either the — sign or a space.
- # Use a variant of the main conversion operation. (See the description of conversion operations below for details.)

Field Size Specification (optional):

The minimal field width, expressed as a decimal integer. `arg` will be printed in a field at least this wide. If `arg` is shorter than field, field will be padded on the side determined by the field adjusting flags. The pad character is normally a space. The pad character is set to zero if the field width is given with a leading 0. For example:

```
printf("%03 d", 2);
```

prints:

```
002
```

with width=3, zero padding. (The zero padding does not imply octal values.)

Precision Specification (optional):

The precision specification identifies the maximum number of characters to be printed from a string or the number of characters to be printed to the right of the decimal point for a long or double.

It consists of a period (.) followed by an optional decimal integer. If the integer is omitted, a precision of 0 is assumed. This is not the same as omitting the precision specification entirely.

An asterisk (*) can also follow the period; in this case, the appropriate `arg` (selected by its corresponding position in the argument list) is used to specify the precision. This argument must be an `int`.

Argument Size Specifications (optional):

- l argument is a long

Standard Libraries Reference

h argument is a halfword (a short)

Conversion Characters (required):

c argument is a single character

d argument printed in signed decimal

e, E argument is printed in signed decimal floating point format ([-] d. ddde+ - dd). One digit appears before the decimal point, and the precision specifies the number of digits to be printed after the decimal point. The only difference between e and E, is that in the latter case, the exponent is introduced by the letter E rather than the letter e.

f argument is printed in signed decimal fixed point format ([-] ddd. dddd). The precision specifies the number of digits to be printed after the decimal point. If precision is missing, 6 digits are output; if precision is 0, no decimal point appears.

g, G Use the f or e format, whichever is smaller. G will cause E to be used instead of e.

o argument is printed in unsigned octal.

s argument is a string. Print until null character or have filled field specified by precision.

u argument printed in unsigned decimal.

x argument printed in unsigned hexadecimal, using the digits 0123456789abcdef

X argument printed in unsigned hexadecimal, using the digits 0123456789ABCDEF

% print a %, no argument used

EXAMPLES

```
/* print text string followed by a newline.
   It contains no format specifiers, so
   requires no arguments. */
printf("simple text\n");
/* print n = 3; %d is the format specifier
   to print a decimal number */
int n;
n = 3;
printf("n = %d", n);
/* print a date and time in the form:
   weekday, month, day, hour:minute */
char *weekday, *month;
int day, hour, min;
printf("%s, %s, %d, %.2d:%.2d", weekday, month,
      day, hour, min);
```

RETURN VALUE

The number of characters printed.

SEE ALSO

`scanf()`, `cprintf()`, `sprintf()`, `vprintf()`

- PUT A CHARACTER INTO A FILE

putc

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int putc(char c, FILE *stream);
```

DESCRIPTION

putc() adds c to stream. putc() is a macro definition for fputc().

Other functions which put characters into a file are: fputc(), fprintf(), vfprintf(), write(), and fwrite(). putch() puts a character onto the screen and putchar() puts a character onto stdout.

RETURN VALUE

The character argument if success; EOF if failure

SEE ALSO

fputc, fputc(), fprintf(), vfprintf(), write(), fwrite(),
putchar

- PUT A CHARACTER ON SCREEN

putch

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
void putch(char c);
```

DESCRIPTION

`putch()` puts a single character to the screen. Unlike `putc()` and `fputc()`, which put a character to a file, or `putchar()`, which puts a character into `stdout`, it is not possible for `putch()` to fail because the screen will always be there.

`putch()` returns nothing.

`cputs()`, `cprintf()`, and `vcprintf()` also send characters to the screen.

SEE ALSO

`putc()`, `fputc()`, `putchar()`, `cputs()`, `cprintf()`, `vcprintf()`

- PUT A CHARACTER TO STDOUT

putchar

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int putchar(char c);
```

DESCRIPTION

`putchar()` is a macro which expands to `fputc(c, stdout)`. `putchar()` sends a single character to `stdout`. The other functions which send data to `stdout` are `puts()` and `printf()`.

RETURN VALUE

The character argument if success; EOF if error.

SEE ALSO

`cputc()`, `putch()`, `fputc()`, `puts()`, `printf()`

- PUT A STRING TO STDOUT

puts

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int puts(char *s);
```

DESCRIPTION

`puts()` puts the characters from string `s` to `stdout` until it reaches a null byte (`'\0'`) signalling the end of the string. The `'\0'` is discarded, and a carriage return (`'\r'`) is written to `stdout`. There are some differences between `puts()` and `cputs()` (put string to console) and `fputs()` (put string to file), so check the descriptions before using them. `printf()` and `putchar()` also send output to `stdout`.

RETURN VALUE

0 if success; EOF if failure.

SEE ALSO

`fputs()`, `cputs()`, `printf()`, `putchar()`

- MOVE I/O POSITION TO START OF FILE

rewind

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
void rewind(FILE *stream) ;
```

DESCRIPTION

`rewind()` sets the I/O position back to the start of the file. The I/O position is moved automatically during I/O, as well as explicitly by `lseek()` and `fseek()`.

A call to `fseek(fd, 0L, 0)` is the same as a call to `rewind(fd)`.

EXAMPLES

```
rewind(fd);
fseek(fd, 0L, 0); /* same as rewind */
```

SEE ALSO

`fseek()`, `lseek()`

• SCAN AND FORMAT INPUT FROM STDIN

scanf**LIBRARY**

stdio

SYNTAX

```
#include <stdio.h>
int scanf(char *format , ...);
```

DESCRIPTION

`scanf()` reads characters in from `stdin` and formats them into data. That data is stored in variables.

For example, if `scanf()` is expecting an integer and reads in the characters "371", it converts it from a character string into the integer value 371, and stores it in a variable pointed to by one of `args`.

`scanf()` can read in different kinds of data in a single call, so it provides an extremely useful and flexible way to get input .

`cscanf()`, `fscanf()`, and `sscanf()` use the same argument conventions as `scanf()` except when specified otherwise in their own descriptions. This description of `format` and `args` applies to all `scanf()` functions.

The first argument to `scanf()` is `format`, a character string telling `scanf` how to format the data it reads. `format` consists of text and conversion specifications, just as in `printf()`. The conversion specifications tell `scanf()` how to format the data that is being read in. Each conversion specification is matched to a field of input. Depending on the conversion specification, that field of input is either fixed length or goes until the next white space character in the input string.

Ordinary text in `format` is a place-holder which is matched against the input text between input fields. If the input text does not match the text in `format`, the `scanf()` is aborted. Format syntax and the conversion specifications are described below.

The remaining arguments to `scanf` are `args`, a variable number of pointers to variables. There should be one `arg` for each conversion specification in `format` (except for the conversion specifications `%*` and `%%`, which are explained below).

CAUTION: The most common mistake when using `scanf()` is to use the variables themselves rather than the pointers to the variables. `scanf()` needs to know the address of the variables it will use to store data, not their present values. Failing to use addresses to store the values will overwrite random memory. See the examples. If `scanf()` runs out of arguments before reaching the end of `s`, arbitrary values from the stack are used as the address of the arguments and overwrite random memory.

White space characters in format are ignored. White space characters are spaces, tabs, or newlines. Include them to make the format string more readable. White space characters in the input stream separate input fields.

Non-white space characters (except for `%`) are matched to the input characters. They must match perfectly or else the `scanf()` is aborted.

`%` indicates the start of a conversion specification.

Conversion specifications begin with `%` and contain in order:

`%` `[*]` `[width]` `[size]`

Assignment Suppression (optional):

If the first character in the conversion specification is an asterisk (`*`), `scanf()` reads the input field until the next white space character. It does not assign the input field to any variable.

Field Width (optional):

`n` where `n` is an integer. `scanf` reads up to `n` characters for this field. If the field is longer than `n` characters, it skips the rest of the field, and continues with the next part of format at the next white space in the input.

Size Specifications (optional):

`l` arg variable for this field is a long.

`h` arg variable for this field is a half-word (a short).

Conversion Characters (terminate conversion specification):

- c interpret input as a single character; `arg` should be a `char*`; white space characters are not skipped in this case. `%1s` will read in the next non-white space character.
- d interpret input as a decimal integer; `arg` should be an `int*`.
- e,E,f,g,G interpret input as a signed decimal floating point number. These operations are all identical.
- o interpret input as an octal integer; `arg` should be an `int*`; input need not start with 0.
- s interpret argument as a character string; `arg` should be a `char*`; a `'\0'` will be added to `arg` after the string is read in.
- u interpret input as unsigned decimal; `arg` should be an `int*`.
- x,X interpret input as a hexadecimal value; `arg` should be an `int*`; input need not start with 0x. These two specifiers are identical.
- [scan the input for the set of characters in the format string following the [up to a terminating]. You can think of the characters enclosed in the brackets as a "scanset." `scanf()` will take as input the maximum string of characters that match the scanset.

When a circumflex (^) appears as the first character in the scanset (e.g. `[^0123456789]`), it acts as a complement operator, and causes all characters not in the set to be matched.

A range of consecutive characters can be represented by separating the first and last members of the range with a hyphen (for example `[A-Za-z]` will match any alphabetic characters, or `[0-9]` will match the set of digits. If the first character in a range is not lexically smaller than the last, then the characters in the scanset stand for themselves. For example, `[9-1]` will match the three characters "9-1."

To include a right bracket (]) in the scanset, it must be the first character in the set. For example, `[)]` will match a single right bracket.

Standard Libraries Reference

arg should be a `char[]` * large enough to hold the data and the terminating `'\0'`, which will be added automatically.

% input should be the character %; input not stored in a variable in arg. No arg is used in the process.

EXAMPLES

```
int n;
scanf("%d ", &n);
/* correct: &n is a pointer to n */
scanf("%d ", n);
/* incorrect: n is the variable itself, so this will
trash random memory */
```

RETURN VALUE

Number of items successfully input.

SEE ALSO

`cscanf()`, `fscanf()`, `sscanf()`, `getchar()`, `gets()`

- SET CONSOLE SCREEN TO ECHO KEYBOARD INPUT

Set_Echo

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
void Set_Echo(int state);
```

DESCRIPTION

Set_Echo() sets the console to echo input from the keyboard to the screen. If state is nonzero, then echo is enabled; if state is zero, echo is disabled.

Echo is enabled by default.

SEE ALSO

Set_Tab(), getch(), getche()

- SET THE CURRENT STDIO WINDOW

setwindow

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
void setwindow(FILE *windowFileVar);
```

DESCRIPTION

The specified FILE variable becomes the "current" window. The functions `Set_Echo`, `Set_Tab`, `gotoxy`, `getxpos`, `getypos`, and `wputs` implicitly apply to the current window.

(Functions which implicitly apply to `stdout`, such as `printf`, or to the console, such as `cprintf`, are not affected by which window is current.)

Initially, `stdout` is the current window. `fopenw` does not make the new window the current window.

- SET TAB WIDTH OF CONSOLE SCREEN

Set_Tab

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
void Set_Tab(int tab_width);
```

DESCRIPTION

Set_Tab() sets the space between tabs on the console screen to tab_width. Set_Tab() does not change the internal representation of a tab, only the effect of outputting a tab on the console.

If tab_width is less than 1, it is set to 1.

SEE ALSO

Set_Echo()

• FORMAT AND MAKE OUTPUT INTO A STRING

sprintf

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int sprintf(char *s, char *format, ...);
```

DESCRIPTION

`sprintf()` is yet another function based on `printf()`. Just as `printf()` formats text and data into a string of characters and then prints it on `stdout`, `sprintf()` formats text and data into a character array. That character array is left containing a string of output. Therefore, the primary action of `sprintf()` is a side effect: changing the contents of string `s`.

The first argument, `s`, is a pointer to a character array large enough to hold the anticipated result. `format` and the optional `args` are interpreted in the usual way by `printf()` (see `printf()` for details) with the result put in `s`.

`sscanf()` is the inverse of `sprintf()`. It scans and formats input from a string.

Care must be taken to ensure that the buffer pointed to by `s` is large enough to hold the converted string. Also note that the string will be null-terminated automatically by `sprintf`.

RETURN VALUE

The number of characters put into the buffer pointed to by `s`, including null terminator, if successful; EOF if an error occurs.

SEE ALSO

`printf()`, `sscanf()`

• SCAN AND FORMAT INPUT FROM A STRING

sscanf**LIBRARY**

stdio

SYNTAX

```
#include <stdio.h>
int sscanf(char *s, char *format, ...);
```

DESCRIPTION

`sscanf()`, like the other `scanf()` functions (`scanf()`, `cscanf()`, and `fscanf()`), does a scan and format of a series of characters. The special feature of `sscanf()` is that it takes that series of characters from its first argument, a string `s`. `s` should terminate in a null (`'\0'`) character. `format` and `args` are interpreted as usual; see `scanf()` for details. Similarly, `sprintf()` does a format and print to a character array in memory.

CAUTION: The most common mistake when using `sscanf()` is to use the variables themselves rather than the pointers to the variables. `sscanf()` needs to know the address of the variables it will use to store data, not their present values. Failing to use addresses to store the values will overwrite random memory. See the examples in `scanf()`. If `sscanf()` runs out of arguments before reaching the end of `s`, arbitrary values from the stack are used as the address of the arguments and overwrite random memory.

RETURN VALUE

Number of items successfully input.

SEE ALSO`scanf()`, `fscanf()`, `cscanf()`, `sprintf()`

- HANDLE EVENT IN A STDIO WINDOW

StdEvent

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
Boolean StdEvent(EventRecord *theEvent);
```

DESCRIPTION

Passed a pointer to an event record, this procedure determines if the event relates to one of the windows created with `fopenw`. If so, it handles the event and returns `TRUE`; otherwise it returns `FALSE`. If your application has its own event loop, you should call `StdEvent` after calling `GetNextEvent`, and handle the event yourself only if `StdEvent` returns `FALSE`.

RETURN VALUE

`TRUE` if event relates to stdio window, `FALSE` otherwise.

• CONFIGURE FONT FOR STDIO

Stdio_config**LIBRARY**

stdio

SYNTAX

```
#include <stdio.h>
void Stdio_config(int font, int size, int style, int
mode);
```

DESCRIPTION

`Stdio_config()` configures standard I/O to handle Macintosh font information. If you want to change the default I/O configuration, you must call this function before doing any screen I/O.

`font` is the number of a Macintosh font. `size` is the point size. `style` is the font style (e.g., bold, underscore, etc.). `mode` is the transfer mode (`srcCopy`, `srcOR`, etc.).

See `FontMgr.h` for symbolic definitions of font names. See `QuickDraw.h` for symbolic definitions of style and transfer mode.

The default values if `Stdio_config()` is not called are `monaco`, 9, 0, 1, which corresponds to Monaco font, 9 point, normal style, source bits ORed.

- INHIBIT STDIO INITIALIZATION

Stdio_MacInit

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
void Stdio_MacInit(Boolean flag);;
```

DESCRIPTION

To prevent stdio functions from calling InitGraf, InitFonts, InitWindows, InitDialogs, TEInit, and InitMenus, and from setting up the stdio menus, etc., call this function with the argument TRUE prior to any stdio calls. This will allow you to use stdio windows without conflicting with your application's own windows and menus.

- GET VERSION OF STDIO LIBRARY

std_ver

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
char *std_ver(void)
```

DESCRIPTION

This returns a pointer to a string containing the version specification of the stdio library. This string is currently "LightspeedC™ Libraries 1.57".

RETURN VALUE

String representation of stdio library version.

- CONVERT A HEXADECIMAL CHARACTER INTO AN INTEGER

toint

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int toint(char c);
```

DESCRIPTION

`toint()` returns the integer corresponding to the hexadecimal value of a character `c`. For example, `toint('2')` returns the int 2, `toint('a')` returns the int 10, `toint('F')` returns the int 15, `toint('r')` returns -1 to indicate an error. `toint()` accepts upper or lower case. Note that `toint()` is not the same as `atoi()` which converts a whole string into an int (as opposed to a character), nor is it the inverse of `toascii()`.

RETURN VALUE

-1 if error; the int that corresponds to the argument if success.

SEE ALSO

`toascii()`, `atoi()`

- CONVERT A CHARACTER TO LOWER CASE

tolower

tolower

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int tolower(char c);

int _tolower(char c);
```

DESCRIPTION

`tolower()` returns the lower case equivalent of an upper case letter `c`. If `c` is not an upper case letter, then `c` is returned unchanged.

`_tolower()` is faster. However it returns a wrong value if its argument is not an upper case letter.

RETURN VALUE

The lower case equivalent of upper case character `C`, or if `c` is not an upper case letter, `c` itself.

SEE ALSO

`toupper()`, `_toupper()`

- CONVERT A CHARACTER TO UPPER CASE

toupper

_toupper

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int toupper(char c);

int _toupper(char c);
```

DESCRIPTION

`toupper()` returns the upper case equivalent of a lower case letter `c`. If `c` is not a lower case letter, `toupper()` returns it unchanged.

`_toupper()` is faster. However, it returns a wrong value if its argument is not a lower case letter.

RETURN VALUE

The upper case equivalent of lower case letter `c`; if `c` is not a lower case letter, `c` itself.

SEE ALSO

`tolower()`

- RETURN CHARACTER C TO INPUT STREAM BUFFER

ungetc

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int ungetc(int c, FILE *stream);
```

DESCRIPTION

`ungetc()` pushes a single character `c` back to the I/O stream. The next read from that stream will get that character first.

`ungetc(s)`, where `s` is an input stream from the keyboard, works by manipulating the `look_full` and `look_ahead` fields of the file descriptor (see `FILE`). When `s` is a file, the character is placed back into the buffer. If the character is different from the last character read and the buffer has been modified by a write, the new character will be written to the file.

RETURN VALUE

`c` if successful; EOF if not. Pushing back EOF places the value of EOF on the stream and returns the value of EOF.

SEE ALSO

`getc()`

• PUSH CHARACTER BACK TO KEYBOARD INPUT STREAM

ungetch

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
char ungetch(char c);
```

DESCRIPTION

ungetch() pushes the character c back to the keyboard input stream.

RETURN VALUE

The character c, if the character could be pushed back; otherwise, EOF.

SEE ALSO

getch(), ungetc()

• VARIABLE FORMAT AND PRINT TO STDOUT

vprintf

vcprintf

vfprintf

vsprintf

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
int vprintf(char *format, char *array);

int vcprintf(char *format, char *array);

int vfprintf(char *format, char *array);

int vsprintf(char *format, char *array)y;
```

DESCRIPTION

The `vprintf` family of routines is entirely analogous in function to the similarly named routines lacking the `v` prefix. That is, `vprintf` performs the same function as `printf`, `vfprintf` as `fprintf`, and so on.

The difference between each pair of routines is that instead of an argument list corresponding to the format specifiers in `format`, the `vprintf` functions take a single array of arguments.

In building this array, you must be sure that each item in it corresponds to the appropriate format specifier in `format`.

RETURN VALUE

The number of characters written, not counting the terminating null character (`'\0'`).

SEE ALSO

`printf()`, `cprintf()`, `fprintf()`, `sprintf()`

- WRITE A STRING INTO THE CURRENT STDIO WINDOW

wputs

LIBRARY

stdio

SYNTAX

```
#include <stdio.h>
void wputs(char *s);
```

DESCRIPTION

This is the same as `cputs`, but output goes to the current window rather than to the console window.

SEE ALSO

`setwindow`

strings

3

Introduction

The `strings` library provides routines to copy, concatenate, and scan strings. All of the routines in this library operate on null terminated C strings. If you need to use these routines on Pascal strings, you can use the `CtoPstr()` and `PtoCstr()` routines which are a part of the MacTraps library. These are their prototypes:

```
char *CtoPstr(char *s);  
char *PtoCstr(char *s);
```

Both of these routines convert the string in place and return a pointer the converted string. Since Pascal strings begin with a length byte, they can be no longer than 255 characters.

If you have the MacHeaders option turned off, be sure to include the file `pascal.h`.

- COPY FIRST N CHARACTERS FROM S2 TO S1

stccpy

LIBRARY

strings

SYNTAX

```
#include <strings.h>
int stccpy(char *s1, char *s2, int n);
```

DESCRIPTION

stccpy() copies n characters from s2 to s1. stccpy() will also stop if it copies a null character ('\0') from s2 to s1. The area pointed to by s1 must be large enough to hold n characters. A '\0' is appended to s1 only if less than n characters could be copied from s2 before reaching the end of s2.

The character pointers are incremented by stccpy(). Therefore, the copy will overlap what was copied before if (s2 < s1 < (s2 + n))

If n is negative or 0, stccpy() does not copy any characters and returns 0.

strcpy() and strncpy() also copy one string to another, but with some small differences.

The destination string is always null terminated.

EXAMPLES

```
A = "ABCD"; B = "EFGHIJ";
stccpy(A,B); /* A = "EFGH" */
```

RETURN VALUE

The number of characters copied if n is positive (including null terminator). 0 if n is negative or zero.

SEE ALSO

strcpy(), strncpy()

- RETURN POSITION OF FIRST CHARACTER IN S NOT IN SET

stcis

LIBRARY

strings

SYNTAX

```
#include <strings.h>
int stc
```

is(char *s, char *set);

DESCRIPTION

stc

is() returns the position of the first character in string s that is not in set. The position of the first character in s is 0, the second character is position 1, etc.

If all characters in s are in set, then stc

is() returns the position of the null character which terminates string s. In effect, this is the length of the string, not counting the null at the end.

set is implemented as a string, but it is logically used as a group of characters with order unimportant.

strspn() is a synonym for stc

is().

RETURN VALUE

The position of the first character in s that is not in set.

SEE ALSO

strspn(), stc

isn(), strcspn(), strpbrk(), strrpbrk()

- RETURN POSITION OF FIRST CHARACTER IN S THAT IS IN SET

stciscn

LIBRARY

strings

SYNTAX

```
#include <strings.h>
int stciscn(char *s, char *set);
```

DESCRIPTION

`stciscn()` finds the first character in string `s` which is a member of string `set`. `stciscn()` returns an `int` which is the place of the first letter in `s` to be in the set (see examples). The first character in the string `s` is in place 0, the second is in place 1, etc.

If no character in `s` is a member of the set, then `stciscn()` returns the place of the null character which terminates string `s`. This is also the length of the string as returned by `strlen()`.

The second argument is called a set for semantic reasons; however, it is implemented simply as a string.

`strcspn()` is a synonym for `stciscn()`.

EXAMPLE

```
stciscn("word", "aeiou")
/* returns 1 (the place of the o in word) */

stciscn("igloo", "aeiou")
/* returns 0 (the place of the i in igloo) */

stciscn("GREEN", "aeiou")
/* returns 5 (the place of the '\0' in GREEN) */
```

RETURN VALUE

The place of the first character in `s` which is in `set`, or the place of the null which terminates `s` if no character in `s` is in `set`.

SEE ALSO

`stciscn()`, `strcspn()`, `stpbrk()`, `strpbrk()`

- RETURN THE LENGTH OF STRING S

strlen

LIBRARY

strings

SYNTAX

```
#include <strings.h>
int strlen(char *s);
```

DESCRIPTION

strlen() simply returns the number of non-null characters in string s.

strlen() is a synonym for strlen().

EXAMPLE

```
strlen("the cat") /* returns 7 */
strlen("") /* returns 0 */
```

RETURN VALUE

The number of non-null characters in s.

SEE ALSO

strlen()

- RETURN POINTER TO FIRST BLANK SPACE CHARACTER IN PTR

stpblk

LIBRARY

strings

SYNTAX

```
#include <strings.h>
char *stpblk(char *ptr);
```

DESCRIPTION

stpblk() finds the first white space character in ptr.

stpblk() does not stop at a '\0' character marking the end of the string, but will continue on through memory until a white space character is found.

RETURN VALUE

p where p is a char* to the first whitespace character in ptr.

SEE ALSO

isspace()

- RETURN POINTER TO FIRST CHARACTER IN S THAT IS IN SET

stpbrk

LIBRARY

strings

SYNTAX

```
#include <strings.h>
char *stpbrk(char *s, char *set);
```

DESCRIPTION

`stpbrk()` finds the first character in string `s` that is in `set`. `stpbrk()` returns a pointer to that first character in `s` (not to the matched character in `set`). It returns `NULL` if no characters match.

Note that `set` is implemented as a `char*`, but it is used logically as a set of characters.

`strpbrk()` is a synonym for `stpbrk()`.

`strcspn()` also finds the first character of a string `s` in a `set`, but it returns an `int` which is the position of the character rather than a pointer to the actual character. `strrpbkr()` returns a pointer to the last character in `s` which appears in `set`.

RETURN VALUE

A pointer to the first character in `s` that is in `set`; `NULL` if no characters in `s` are in `set`.

SEE ALSO

`strpbrk()`, `strcspn()`, `strrpbkr()`

- RETURN POINTER TO FIRST OCCURRENCE OF C IN S

stpchr

LIBRARY

strings

SYNTAX

```
#include <strings.h>
char *stpchr(char *s, char c);
```

DESCRIPTION

`stpchr()` returns a character pointer to the first occurrence of character `c` in string `s`. If `c` is not in `s`, `stpchr()` returns `NULL`. `stpchr()` will not search past the null character (`'\0'`) which ends string `s`. If `c` is equal to `'\0'`, it is considered to match the `'\0'` which terminates string `s` and a pointer to that position is returned.

`strchr()` is a synonym for `stpchr()`.

`strpos()` also finds a character `c` in a string `s`, but `strpos()` returns an `int` which is the position of `c` in `s` rather than a pointer to that character.

`strrpos()` and `strrchr()` return the position of the last `c` in `s` and a pointer to the last `c` in `s`, respectively.

RETURN VALUE

`NULL` if `c` is not in `s`; otherwise, a pointer to the first occurrence of `c` in `s`.

SEE ALSO

`strpos()`, `strrpos()`, `strrchr()`, `strchr()`

- COPY S2 INTO S1

strcpy

LIBRARY

strings

SYNTAX

```
#include <strings.h>
char *strcpy(char *s1, char *s2);
```

DESCRIPTION

`strcpy()` copies the entire string `s2` into `s1`. `strcpy()` uses `(strlen(s2) + 1)` as the third argument to `strncpy()` to ensure that `s2` is copied in its entirety.

See `strncpy()` and `strlen()` for details of their functions.

Be sure that the character array pointed to by `s1` is sufficiently large.

`strcpy()` is synonymous with `strcpy()`.

RETURN VALUE

A pointer to `s1`.

SEE ALSO

`strncpy()`, `strcpy()`, `strcpy()`

- APPEND S2 TO S1

strcat

LIBRARY

strings

SYNTAX

```
#include <strings.h>
char *strcat(char *s1, char *s2);
```

DESCRIPTION

`strcat()` calls `strncat()` to append the entire string `s2` to the string pointed to by `s1`.

`strcat()` uses `(strlen(s2) + 1)` as the third argument to `strncat()` to ensure that `s2` is appended in its entirety. Be sure that there is enough space in the character array after the end of the string `s1` to allow `s2` to be appended.

The null character which ended string `s1` is overwritten by the first character from `s2`.

RETURN VALUE

`s1` with `s2` appended to it.

SEE ALSO

`strlen()`, `strncat()`

- RETURN POINTER TO FIRST OCCURRENCE OF C IN S

strchr

LIBRARY

strings

SYNTAX

```
#include <strings.h>
char *strchr(char *s, char c);
```

DESCRIPTION

`strchr()` returns a character pointer to the first occurrence of character `c` in string `s`. If `c` is not in `s`, `strchr()` returns `NULL`. `strchr()` will not search past the null character (`'\0'`) which ends string `s`.

If `c` is equal to `'\0'`, it is considered to match the `'\0'` which terminates string `s` and a pointer to that position is returned.

`stpchr()` is a synonym for `strchr()`.

`strpos()` also finds a character `c` in a string `s`, but `strpos()` returns an `int` which is the position of `c` in `s` rather than a pointer to that character.

`strrpos()` and `strrchr()` return the position of the last `c` in `s` and a pointer to the last `c` in `s` respectively.

RETURN VALUE

`NULL` if `c` is not in `s`; otherwise, a pointer to first occurrence of `c` in `s`.

SEE ALSO

`strpos()`, `strrpos()`, `strrchr()`, `stpchr()`

• LEXICALLY COMPARE S1 AND S2

strcmp

LIBRARY

strings

SYNTAX

```
#include <strings.h>
int strcmp(char *s1, char *s2);
```

DESCRIPTION

`strcmp()` calls `strncmp()` to compare the length of `s1` to the length of `s2`. `strcmp()` uses `(strlen(s1) + 1)` as the third argument to `strncmp()` to ensure that `s1` is compared to `s2` in its entirety.

Note that if `s1` is a substring of `s2`, then `strcmp()` will return a number greater than 0 because the last characters compared will be the terminating null (`'\0'`) character of `s1` against some character in `s2`.

See `strncmp()` and `strlen()` for details of their functions.

`stscmp()` is a synonym for `strcmp()`.

RETURN VALUE

0 if string `s1` is the same as string `s2`; positive if `s1 > s2`; negative if `s1 < s2`.

SEE ALSO

`strncmp()`, `strlen()`, `stscmp()`

- COPY S2 INTO S1

strcpy

LIBRARY

strings

SYNTAX

```
#include <strings.h>
char *strcpy(char *s1, char *s2);
```

DESCRIPTION

`strcpy()` calls `strncpy()` to copy the entire string `s2` into `s1`. `strcpy()` uses `(strlen(s2) + 1)` as the third argument to `strncpy()` to ensure that `s2` is copied in its entirety.

See `strncpy()` and `strlen()` for details of their functions.

Be sure that the character array pointed to by `s1` is large enough to hold the whole of `s2`.

`stpcpy()` is a synonym for `strcpy()`.

RETURN VALUE

A pointer to `s1`.

SEE ALSO

`strncpy()`, `stccpy()`, `stpcpy()`

- RETURN POSITION OF FIRST CHARACTER IN *s* THAT IS IN SET

strcspn

LIBRARY

strings

SYNTAX

```
#include <strings.h>
int strcspn(char *s, char *set);
```

DESCRIPTION

`strcspn()` finds the first character in string *s* which is a member of string *set*. `strcspn()` returns an `int` which is the place of the first letter in *s* to be in the set (see examples). The first character in the string *s* is in place 0, the second is in place 1, etc.

If no character in *s* is a member of the set, then `strcspn()` returns the place of the null character which terminates string *s*. This is also the length of the string as returned by `strlen()`.

The second argument is called *set* for semantic reasons, however it is implemented simply as a string.

`stcispn()` is a synonym for `strcspn()`.

EXAMPLES

```
strcspn("word", "aeiou")
/* returns 1 (the place of the o in word) */

strcspn("igloo", "aeiou")
/* returns 0 (the place of the i in igloo) */

strcspn("GREEN", "aeiou")
/* returns 5 (the place of the '\0' in GREEN) */
```

RETURN VALUE

The place of the first character in *s* which is in *set*. If no character in *s* is in *set*, then the return value is the position of the null that terminates *s*.

SEE ALSO

`stcispn()`, `stcispn()`, `strspn()`, `stpbrk()`, `strpbrk()`

- RETURN THE LENGTH OF STRING S

strlen

LIBRARY

strings

SYNTAX

```
#include <strings.h>
int strlen(char *s);
```

DESCRIPTION

strlen() simply returns the number of non-null characters in string s.

stclen() is a synonym for strlen().

RETURN VALUE

The number of non-null characters in s.

EXAMPLES

```
strlen("the cat") /* returns 7 */
strlen("") /* returns 0 */
```

SEE ALSO

stclen()

- APPEND UP TO N CHARACTERS FROM S2 TO S1

strncat

LIBRARY

strings

SYNTAX

```
#include <strings.h>
char *strncat(char *s1, char *s2, int n);
```

DESCRIPTION

`strncat()` appends `s2` to `s1` until it has appended `n` characters or it has reached the end of `s2`. The characters are added to `s1` starting at the first null character (`'\0'`) of `s1`. That `'\0'` is replaced by the first character of `s2`.

If `n` characters are appended without reaching the end of `s2`, `strncat()` adds a terminating null character (`'\0'`).

If `n` is negative or zero, `strncat()` returns `s1` unchanged.

RETURN VALUE

`s1` where `s1` is the first argument with the contents of `s2` appended to it.

SEE ALSO

`strcat()`

- COMPARE UP TO FIRST N CHARACTERS OF S1 AND S2

strncmp

LIBRARY

strings

SYNTAX

```
#include <strings.h>
int strncmp(char *s1, char *s2, int n);
```

DESCRIPTION

strncmp() compares s1 and s2 up to a limit of n, and returns an integer indicating whether s1 is equal to, less than, or greater than s2.

strcmp() follows the same rules except that it compares until it reaches the end of one or both strings.

RETURN VALUE

0 if string s1 is the same as string s2 in the first n characters; positive if s1 > s2 in the first n characters; negative if s2 > s1 in the first n characters.

SEE ALSO

strcmp(), stscmp()

- COPY UP TO N CHARACTERS FROM S2 TO S1

strncpy

LIBRARY

strings

SYNTAX

```
#include <strings.h>
char *strncpy(char *s1, char *s2, int n);
```

DESCRIPTION

`strncpy()` copies characters from `s2` to `s1` until either `n` characters have been copied or it has reached the null character (`'\0'`) which marks the end of `s2`. If the end of `s2` is reached first, $(n - \text{strlen}(s2))$ null characters are appended as padding. No null characters are appended if `n` characters from `s2` are added without reaching a `'\0'` in `s2`.

Thus, to reiterate, the first `n` characters of `s1` will be written over, either by the first `n` characters from `s2`, or if `s2` ends early, `s2` plus enough null character padding.

If `n` is negative or zero, `s1` is returned unchanged. `strcpy()` also copies `n` characters from `s2` to `s1`.

`strcpy()` handles null padding differently, and returns the number of characters copied instead of a pointer to `s1`.

`strcpy()` copies `s2`, and nothing but `s2`, to `s1`.

RETURN VALUE

A pointer to `s1`.

SEE ALSO

`strcpy()`, `strcpy()`

- RETURN POINTER TO FIRST CHARACTER IN S THAT IS IN SET

strpbrk

LIBRARY

strings

SYNTAX

```
#include <strings.h>
char *strpbrk(char *s, char *set);
```

DESCRIPTION

`strpbrk()` finds the first character in string `s` that is in `set`. `strpbrk()` returns a pointer to that first character in `s` (not to the matched character in `set`). It returns `NULL` if no characters match.

`stpbrk()` is a synonym for `strpbrk()`.

`strcspn()` also finds the first character of a string `s` in a set, but it returns an `int` which is the position of the character rather than a pointer to the actual character. `strpbrk()` returns a pointer to the last character in `s` which appears in `set`.

RETURN VALUE

A pointer to the first character in `s` that is in `set`; `NULL` if no characters in `s` are in `set`.

SEE ALSO

`stpbrk()`, `strcspn()`, `strpbrk()`

- RETURN POSITION OF THE FIRST CHARACTER C IN STRING S

strpos

LIBRARY

strings

SYNTAX

```
#include <strings.h>
int strpos(char *s, char c);
```

DESCRIPTION

`strpos()` searches `s` from beginning to end for the character `c`. `strpos()` returns the first position of `c` in `s`, with the first character in `s` being at position 0. `strpos()` returns -1 if `c` is not in `s`.

`strchr()` also finds a character `c` in a string `s`, but returns a pointer to the first such character.

`strrpos()` and `strrchr()` return the position of the last `c` and a pointer to the last `c` in `s` respectively.

RETURN VALUE

The first position of `c` in `s`; -1 if `c` is not in `s` or if `c` is `'\0'`.

SEE ALSO

`strchr()`, `strrchr()`, `strrpos()`

- RETURN POINTER TO LAST OCCURRENCE OF C IN S

strrchr

LIBRARY

strings

SYNTAX

```
#include <strings.h>
char *strrchr(char *s, char c);
```

DESCRIPTION

`strrchr()` searches string `s` from beginning to end for character `c`. `strrchr()` returns a pointer to the last occurrence of `c` that it finds.

If `c` is the null character (`'\0'`), `strrchr()` returns a pointer to the null character which terminates `s`. `strrchr()` returns `NULL` if `c` is not in `s`. `strpos()` also finds the last occurrence of a character `c` in a string `s`, but returns its position as an integer rather than a pointer to it.

`strpos()` and `strchr()` return the position of the first `c` and a pointer to the first `c` in `s` respectively.

RETURN VALUE

A pointer to the last occurrence of `c` in `s`; `NULL` if `c` is not in `s`.

SEE ALSO

`strrchr()`, `strpos()`, `strchr()`

• RETURN POINTER TO LAST CHARACTER IN STRING *s* THAT IS IN SET

strrpbrk

LIBRARY

strings

SYNTAX

```
#include <strings.h>
char *strrpbrk(char *s, char *set);
```

DESCRIPTION

`strrpbrk()` finds the last character in *s* that is in *set*, and returns a pointer to that character. The pointer is to the character in *s*, not the character it matches in *set*.

Note that while *set* is implemented as a character string, the order of characters in it does not matter. A lower case character does not match its equivalent upper case character.

`strpbrk()` returns the pointer to the first character in *s* that is in *set*.

RETURN VALUE

A character pointer to the last character in *s* that is in *set*; `NULL` if no character in *s* is in *set*.

SEE ALSO

`strpbrk()`, `strspn()`, `strcspn()`, `strcspn()`

- RETURN POSITION OF THE LAST CHARACTER C IN STRING S

strrpos

LIBRARY

strings

SYNTAX

```
#include <strings.h>
int strrpos(char *s, char c);
```

DESCRIPTION

`strrpos()` searches string `s` from beginning to end for character `c`. `strrpos()` returns the last position of `c`, with the first character of `s` being at position 0. `strrpos()` returns -1 if `c` is not in `s`.

If `c` is the null character (`'\0'`), `strrpos()` returns the position of the `'\0'` which terminates string `s`.

`strrchr()` also finds a character `c` within a string `s`, but returns a pointer to the last such character.

`strpos()` and `strchr()` return the position of the first `c` in `s` and a pointer to the first `c` in `s` respectively.

RETURN VALUE

The position of the last `c` in `s`; -1 if `c` is not in `s`.

SEE ALSO

`strrchr()`, `strpos()`, `strchr()`

•RETURN POSITION OF FIRST CHARACTER IN S NOT IN SET

strspn

LIBRARY

strings

SYNTAX

```
#include <strings.h>
int strspn(char *s, char *set);
```

DESCRIPTION

`strspn()` returns the position of the first character in string `s` that is not in `set`. The position of the first character in `s` is 0, the second character is position 1, etc.

If all characters in `s` are in `set`, then `strspn()` returns the position of the null character which terminates string `s`. `set` is implemented as a string, but it is logically used as a set of characters with order unimportant.

`stcisc()` is a synonym for `strspn()`.

RETURN VALUE

The position of the first character in `s` that is not in `set`.

SEE ALSO

`stcisc()`, `stciscn()`, `strcspn()`, `strpbrk()`, `strrpbrk()`

•LEXICALLY COMPARE S1 AND S2

stscmp**LIBRARY**

string

SYNTAX

```
#include <strings.h>
int stscmp(char *s1, char *s2);
```

DESCRIPTION

`stscmp()` calls `strncmp()` to compare the entire length of `s1` to the length of `s2`. `stscmp()` uses `(strlen(s1) + 1)` as the third argument to `strncmp()` to ensure that `s1` is compared to `s2` in its entirety.

Note that if `s1` is a substring of `s2`, then `stscmp()` will return a number greater than 0 because the last characters compared will be the terminating null character (`'\0'`) of `s1` against some character in `s2`.

See `strncmp()` and `strlen()` for details of their functions.

`strcmp()` is a synonym for `stscmp()`.

RETURN VALUE

0 if string `s1` is the same as string `s2`; positive if `s1 > s2`; negative if `s1 < s2`.

SEE ALSO

`strncmp()`, `strlen()`, `strcmp()`

Standard Libraries Reference

math 4

Introduction

The math libraries supplied with THINK C provide standard mathematic calculation routines as specified in the Second Edition of *The C Programming Language*. There are a few things you should keep in mind when you use the math libraries with THINK C.

- The global variable `errno` is defined in the `stdio` library. If you're not using the `stdio` library in your program, you should declare `errno` as a global variable of type `int` somewhere in your program; the declaration should not be `static`.
- The values `EDOM` and `ERANGE` are declared as an enums in `<math.h>`.
- You should include `<math.h>` in any source file where you use the math routines to ensure that the functions are properly declared. If you don't, you will get wrong answers, and your program may not behave correctly.

68881 Coprocessor Support

The math libraries supplied with THINK C come in three variations: `Math`, `MathHybrid`, and `Math881`.

Use `Math` when you are *not* using the 68881 code-generation capability of THINK C.

Use `MathHybrid` when you are using the 68881 code generation, but you want to use the SANE functions built into the Macintosh ROM for maximum precision. The tradeoff for using `MathHybrid` is that you do not derive the full performance benefits from the 68881 code generation.

Use `Math881` when you are using the 68881 code generation. The routines in this library access the 68881 directly for maximum performance. The tradeoff for using `Math881` is that the transcendental functions (trigonometric and log/exponential functions) will not be as accurate as their SANE counterparts. The benefit for using `Math881` is that the functions in this library are about *two hundred* times faster than the functions in `MathHybrid`.

You can access the math functions by adding the appropriate library to your project file, and including `<math.h>` in source files where you need to access the library functions.

Error Checking

The Math and MathHybrid libraries do standard error checking. The routines in these libraries test their input arguments. If the result would be undefined, the global variable `errno` is set to `EDOM` and the routines return the largest representable double-precision value.

Note: In cases where the result would be an infinitely large negative number, the result is the largest double-precision value with a negative sign.

The routines also check their results before they return. If the result is not representable as a double-precision number, `errno` will be set to `ERANGE`.

If you want to disable error-checking to increase performance, you undefine the symbol `_ERRORCHECK_` and recompile the appropriate math library. See the next section for details.

Preprocessor Symbols

There are two preprocessor symbols that you should define before including `<math.h>` in your source files. The easiest way to do this is to create a header file to hold these definitions, and include it before you include `<math.h>`.

The proper syntax for defining these symbols is:

```
#define _ERRORCHECK_      /* for standard error checking */
#define _MC68881_         /* if you're using 68881 code generation */
```

Note: If these symbols are not defined, you may get linker errors.

If `_ERRORCHECK_` is defined, the math routines do standard error checking as described above). If you forget to define this symbol, you're likely to see the error message:

```
undefined: errno (math)
```

By default, Math and MathHybrid perform error checking, so `_ERRORCHECK_` must be defined to use these libraries.

If `_MC68881_` is defined, the libraries are made aware that you are using THINK C's code-generation support for the 68881. **This is important if you are using MathHybrid or Math881.** If you're using the 68881 code generation, be sure to define `_MC68881_`. Conversely, it is very important that `_MC68881_` is undefined if you are not using the math coprocessor support provided by the THINK C compiler.

- ABSOLUTE VALUE OF INTEGER

abs**LIBRARY**

math, MathHybrid, Math881

SYNTAX

```
int abs(int i);
```

DESCRIPTION

abs returns the absolute value of its argument, which is an integer.

RETURN VALUE

$|i|$ for all i .

•TRIGONOMETRIC ARCCOSINE

acos

LIBRARY

math, MathHybrid, Math881

SYNTAX

```
double acos(double x);
```

DESCRIPTION

`acos` returns the inverse cosine of the argument `x` for `x` in the range of -1 to +1.

RETURN VALUE

$\cos^{-1} x$ in the range 0 to π . If $|x|$ is greater than one, zero is returned and `errno` is set to `EDOM`.

•TRIGONOMETRIC ARCSINE

asin**LIBRARY**

math, MathHybrid, Math881

SYNTAX`double asin(double x);`**DESCRIPTION**

`asin` returns the inverse sine of the argument `x` for `x` in the range of -1 to +1.

RETURN VALUE

$\sin^{-1} x$ in the range $-\pi/2$ to $+\pi/2$. If $|x|$ is greater than one, zero is returned and `errno` is set to `EDOM`.

- TRIGONOMETRIC ARCTANGENT

atan

LIBRARY

math, MathHybrid, Math881

SYNTAX

```
double atan(double x);
```

DESCRIPTION

atan returns the inverse tangent of the argument x.

RETURN VALUE

$\tan^{-1} x$ in the range $-\pi$ to $+\pi$.

• ARCTANGENT OF A QUOTIENT

atan2**LIBRARY**

math, MathHybrid, Math881

SYNTAX`double atan2(double y, double x);`**DESCRIPTION**

`atan2` returns the inverse tangent of y/x , with the same sign as y . If both x and y are zero, the result is zero, and `errno` is set to `EDOM`.

RETURN VALUE

$\tan^{-1}(y/x)$ in the range $-\pi$ to $+\pi$.

- ROUND UP TO NEAREST INTEGER

ceil

LIBRARY

math, MathHybrid, Math881

SYNTAX

```
double ceil(double x);
```

DESCRIPTION

`ceil` rounds the argument to the next higher integer; for example, `ceil(1.1)` will be 2, and `ceil(4.9)` is 5.

RETURN VALUE

`x`, rounded up to the next integer.

- TRIGONOMETRIC COSINE

COS**LIBRARY**

math, MathHybrid, Math881

SYNTAX

```
double cos(double x);
```

DESCRIPTION

cos returns the cosine of the argument x .

RETURN VALUE

cos x .

- HYPERBOLIC COSINE

cosh

LIBRARY

math, MathHybrid, Math881

SYNTAX

```
double cosh(double x);
```

DESCRIPTION

`cosh` returns the hyperbolic cosine of the argument `x`.

RETURN VALUE

`cosh x`.

- BASE-E POWER OF X

exp**LIBRARY**

math, MathHybrid, Math881

SYNTAX

```
double exp(double x);
```

DESCRIPTION

exp returns E (the base of natural logarithms) to the Xth power.

RETURN VALUE

e^x . If the argument is so large that the result is not representable as a double, the largest possible double is returned and errno is set to ERANGE.

• FLOATING-POINT ABSOLUTE VALUE

fabs

LIBRARY

math, MathHybrid, Math881

SYNTAX

```
double fabs(double x);
```

DESCRIPTION

`fabs` returns the absolute value of its floating-point absolute value.

RETURN VALUE

$|x|$ for all values of x .

- ROUND DOWN TO NEAREST INTEGER

floor

LIBRARY

math, MathHybrid, Math881

SYNTAX

```
double floor(double x);
```

DESCRIPTION

`floor` rounds the argument to the next floor integer; for example, `floor(1.1)` will be 1, and `floor(4.9)` is 4.

RETURN VALUE

`x`, rounded down to the next integer.

• FLOATING-POINT REMAINDER

fmod

LIBRARY	math, MathHybrid, Math881
SYNTAX	<code>double fmod(double x, double y);</code>
DESCRIPTION	<code>fmod</code> computes the remainder of x/y , with the same sign as x . If y is zero, <code>errno</code> is set to <code>EDOM</code> .
RETURN VALUE	remainder of x/y , or <code>max</code> if y is zero.

•SPLIT INTO FRACTION AND EXPONENT

frexp**LIBRARY**

math, MathHybrid, Math881

SYNTAX

```
double frexp(double x, int *exp);
```

DESCRIPTION

`frexp` splits `x` into a mantissa and a power of 2.

RETURN VALUE

The mantissa of `x`, in the range of 0.5 to 1. The exponent is stored in `*exp`. If `x` is zero, the function result and `*exp` will be zero.

SEE ALSO

`ldexp()`, which is the inverse function of `frexp()`.

•ABSOLUTE VALUE OF LONG

labs

LIBRARY

math, MathHybrid, Math881

SYNTAX

```
long labs(long l);
```

DESCRIPTION

`labs` returns the absolute value of its argument, which is a long integer.

RETURN VALUE

`|l|` for all `l`.

•JOIN FRACTION AND EXPONENT

ldexp

LIBRARY

math, MathHybrid, Math881

SYNTAX

```
double ldexp(double x, int exp);
```

DESCRIPTION

ldexp raises 2 to the exp power and multiplies x by the result.

RETURN VALUE

$x * 2^{\text{exp}}$.

SEE ALSO

frexp(), which is the inverse function of ldexp().

- BASE-E LOGARITHM

log

LIBRARY

math, MathHybrid, Math881

SYNTAX

```
double log(double x);
```

DESCRIPTION

`log` returns the natural logarithm of its argument.

RETURN VALUE

$\ln x$ for all x greater than zero. If x is zero or negative, `errno` is set to `EDOM` and the negative of the largest representable double is returned.

- BASE-10 LOGARITHM

log10

LIBRARY

math, MathHybrid, Math881

SYNTAX

```
double log10(double x);
```

DESCRIPTION

log10 returns the natural logarithm of its argument.

RETURN VALUE

$\log_{10} x$ for all x greater than zero. If x is zero or negative, `errno` is set to `EDOM` and the negative of the largest representable double is returned.

- SPLIT INTO INTEGER AND FRACTION

modf

LIBRARY

math, MathHybrid, Math881

SYNTAX

```
double modf(double x, double *ip);
```

DESCRIPTION

`modf` splits `x` into integral and fractional parts; both parts carry the same sign as `x`. The integral part is stored in `*ip`.

RETURN VALUE

the fractional part of `x` is returned.

- POWER FUNCTION

pow**LIBRARY**

math, MathHybrid, Math881

SYNTAX

```
double pow(double x, double y);
```

DESCRIPTION

`pow` returns the first argument raised to the power of the second argument.

RETURN VALUE

x^y for all x greater than zero. If x is zero and y is less than or equal to zero, or if x is negative and y is non-integral, `errno` will be set to `EDOM`.

Standard Libraries Reference

•RANDOM INTEGER

rand

LIBRARY

math, MathHybrid, Math881

SYNTAX

```
int rand();
```

DESCRIPTION

`rand` returns a random integer.

- TRIGONOMETRIC SINE

sin**LIBRARY**

math, MathHybrid, Math881

SYNTAX

```
double sin(double x);
```

DESCRIPTION

sin returns the sine of the argument x.

RETURN VALUE

sin x.

- HYPERBOLIC SINE

sinh

LIBRARY

math, MathHybrid, Math881

SYNTAX

```
double sinh(double x);
```

DESCRIPTION

`sinh` returns the hyperbolic sin of the argument `x`.

RETURN VALUE

`sinh x`.

- SQUARE ROOT

sqrt**LIBRARY**

math, MathHybrid, Math881

SYNTAX

```
double sqrt(double x);
```

DESCRIPTION

`sqrt` returns the square root of its argument.

RETURN VALUE

`x` for all `x` greater than or equal to zero. If `x` is less than zero, `errno` is set to `EDOM` and zero is returned.

•SET UP RANDOM NUMBER GENERATOR

srand

LIBRARY

math, MathHybrid, Math881

SYNTAX

```
int srand(unsigned int seed);
```

DESCRIPTION

srand sets the seed for rand() to "seed".

•TRIGONOMETRIC TANGENT

tan**LIBRARY**

math, MathHybrid, Math881

SYNTAX

```
double tan(double x);
```

DESCRIPTION

`tan` returns the cosine of the argument `x`.

RETURN VALUE

`tan x`. If `x` is an odd multiple of $\pi/2$, `errno` will be set to `EDOM` and the return value will be `max`.

• HYPERBOLIC TANGENT

tanh

LIBRARY

math, MathHybrid, Math881

SYNTAX

```
double tanh(double x);
```

DESCRIPTION

`tanh` returns the hyperbolic tangent of the argument `x`.

RETURN VALUE

`tanh x`.



storage 5

Introduction

The storage library provides high level memory management functions found in most C environments. Note that these routines call the Macintosh memory manager directly.

• ALLOCATE AND CLEAR A BLOCK OF MEMORY

calloc

LIBRARY

storage

SYNTAX

```
#include <storage.h>
char *calloc(unsigned n, unsigned s);
```

DESCRIPTION

`calloc()` dynamically allocates a block of $n*s$ bytes of memory, and clears (zeroes) the block. n is the number of items desired, and s is the size of each item. This routine uses the standard Macintosh memory manager to perform its function.

There are a half dozen different functions to dynamically allocate memory. See the tables in the introduction to this section for a comparison of the functions.

Use the `sizeof` operator to find the size of the item to assure portability (see the example). Although the block will be properly aligned, you should cast the pointer returned by `calloc()` so that it is guaranteed to be formatted as a pointer to the desired type of item.

`cfree()` or `free()` deallocates a block of memory that was allocated by `calloc()`.

EXAMPLES

```
typedef struct
{
    int day, month, year;
    int hours[24];
} date;
char *calloc();
date *newmonth;
if ( (newmonth = (date *) calloc(31, sizeof(date)))
    != NULL )
    { /* do work */ }
```

RETURN VALUE

A pointer to the newly allocated block; NULL if insufficient memory is available.

SEE ALSO

- FREE A BLOCK OF MEMORY

cfree

LIBRARY

storage

SYNTAX

```
#include <storage.h>
int cfree(char *cp);
```

DESCRIPTION

`cfree()` releases a block of memory which was dynamically allocated by `calloc()` or `malloc()`.

The argument, `cp`, is a pointer to an allocated block `cp` must have been allocated by `calloc()` or `malloc()`. If `cp` was not so allocated, anything can happen. There are several other library functions which free dynamically allocated memory. A table in the introduction to this section matches these deallocating functions to the corresponding allocating function(s).

RETURN VALUE

0 if successful; -1 if an error occurs.

SEE ALSO

`calloc()`, `malloc()`, `free()`, `rlsmem()`, `rlsml()`

- CLEAR AND ALLOCATE A LONG BLOCK OF MEMORY

clalloc

LIBRARY

storage

SYNTAX

```
#include <storage.h>
char *clalloc(unsigned long n, unsigned long s);
```

DESCRIPTION

`clalloc()` is yet another function to dynamically allocate memory. `clalloc()` is the same as `calloc()` except that its arguments are longs rather than integers. `clalloc()` differs from `malloc()` in that `clalloc()` clears the block of memory.

In fact, there are a half dozen different functions to dynamically allocate memory. See the tables in the introduction to this section for a comparison of the functions. Be sure that both arguments in a call to `clalloc()` are longs; this usually requires casting the result of the `sizeof` operator, as illustrated in the example `free()` and `cfree()` free memory allocated by `clalloc()`.

EXAMPLES

```
long block_ct;
char c, *c_ptr;
...
c_ptr = clalloc(block_ct, (long) (sizeof(c)));
```

RETURN VALUE

A pointer to the newly allocated block; `NULL` if there is insufficient memory.

SEE ALSO

`calloc()`, `malloc()`, `malloc()`, `cfree()`, `free()`

- RELEASE WHOLE BLOCK OF MEMORY

free

LIBRARY

storage

SYNTAX

```
#include <storage.h>
int free(char *cp);
```

DESCRIPTION

`free()` releases a block of memory which was dynamically allocated by `calloc()`, `malloc()`, `clalloc()` or `mlalloc()`. `free()` automatically knows how much memory to free. It just needs `cp`, the pointer to the start of the block. `cp` must have been a pointer that was returned by `calloc()`, `malloc()`, `clalloc()` or `mlalloc()`. The whole block is freed.

RETURN VALUE

0 if successful; -1 if an error occurs.

SEE ALSO

`calloc()`, `malloc()`, `cfree()`

- ALLOCATE A BLOCK OF MEMORY

malloc

LIBRARY

storage

SYNTAX

```
#include <storage.h>
char *malloc(unsigned n);
```

DESCRIPTION

`malloc()` dynamically allocates `n` bytes of memory. Unlike `calloc()`, it does not clear the block of memory. `malloc()` returns a pointer to the block of memory it has allocated. `free()` and `cfree()` free memory allocated by `malloc()`.

There are actually several different functions to allocate memory. They are described together in the introduction to this section as well as separately by name.

RETURN VALUE

A pointer to the block if success; or `NULL` if failure.

SEE ALSO

`free()`, `cfree()`, `calloc()`, `clalloc()`, `mlalloc()`

- ALLOCATE A LONG BLOCK OF MEMORY

malloc

LIBRARY

storage

SYNTAX

```
#include <storage.h>
char *malloc(unsigned long n);
```

DESCRIPTION

`malloc()` dynamically allocates `n` bytes of memory. `n` is a long, so `malloc()` can allocate a long block with one call. Unlike `calloc()`, `malloc` does not clear that block of memory. `malloc()` returns a pointer to the block of memory it has allocated. `cfree()` and `free()` deallocate as they do with `malloc()`.

There are actually several different functions to allocate memory. They are described together in the introduction to this section as well as separately by name.

RETURN VALUE

A pointer to the block if success; `NULL` if failure.

SEE ALSO

`calloc()`, `calloc()`, `malloc()`, `cfree()`, `free()`



unix 6

Introduction

The `unix` library provides some functions found in UNIX environments. Not all of the routines are meaningful in the Macintosh environment, but they are included for compatibility.

Note that the `setjmp()` and `longjmp()` functions are actually in the library `setjmp.lib`.

•EXIT AND CALL SYSTEM DEBUGGER

abort

LIBRARY

unix

SYNTAX

```
#include <unix.h>
void abort(void);
```

DESCRIPTION

Exits and calls the system debugger if it is loaded. Use `exit()` if you do not want the system debugger to be called. This function does not return or close files.

SEE ALSO

`exit()`, `_exit()`

- CONVERT STRING TO A NUMBER

atoi

atol

atof

LIBRARY

unix

SYNTAX

```
#include <unix.h>
int atoi(char *s);

long atol(char *s);

double atof(char *s);
```

DESCRIPTION

These functions skip over leading white space (`isspace(*s)` is true) and convert the string `s` to the target type.

<code>atoi(s)</code>	Converts <code>s</code> to an integer.
<code>atol(s)</code>	Converts <code>s</code> to a long.
<code>atof(s)</code>	Converts <code>s</code> to a floating point number.

A leading minus sign (-) indicates a negative number.

RETURN VALUE

The converted value of the string.

SEE ALSO

`strtol()`, `strtol()`, `strtol()`, `strtol()`, `strtol()`,
`strtol()`

- CONCATENATE STRING WITH PROCESS ID NUMBER

cgetpid

LIBRARY

unix

SYNTAX

```
#include <unix.h>
char *cgetpid(char *str);
```

DESCRIPTION

`cgetpid()` returns the user's string (`str`) concatenated with the process id number expressed as a five digit number with leading zeros. For example, if the process id = 1, then

```
printf("%s",cgetpid("Process"))
```

will print `Process00001`.

RETURN VALUE

A pointer to a string containing `str` concatenated with the process id as returned by `getpid()`.

SEE ALSO

`getpid()`, `setpid()`

•DRAW A CIRCLE

circle**LIBRARY**

unix

SYNTAX

```
#include <unix.h>
void circle (int x, int y, int r);
```

DESCRIPTION

circle() draws a circle with center x,y and a radius r.

The Macintosh coordinate system is defined with 0,0 at the top left of the screen, and 512,342 at the bottom right. Coordinates are given in pixels. The radius is measured in pixels, as are the center coordinates.

SEE ALSO

gotoxy(), label(), line(), move(), point()

- CLOSE FILE NUMBER FN

close

LIBRARY

unix

SYNTAX

```
#include <unix.h>
int close(int fn);
```

DESCRIPTION

Closes the file whose file descriptor number is `fn`. The file descriptor number is the number returned by `open()` when the file was last opened.

`exit()` automatically closes all open files. `fclose()` closes a file when given a file descriptor number.

RETURN VALUE

0 if successful; -1 if an error occurs.

SEE ALSO

`open()`, `fclose()`, `_closeall()`

- DRAW LINE FROM CURSOR TO X,Y

cont

LIBRARY

unix

SYNTAX

```
#include <unix.h>
void cont(int x, int y);
```

DESCRIPTION

cont () draws a line from the current screen position to position x,y . The Macintosh coordinate system is defined with 0,0 at the top left of the screen, and 512,342 at the bottom right.

SEE ALSO

circle (), gotoxy (), label (), line (), move (), point ()

- CREATE A NEW FILE CALLED FILENAME

creat

LIBRARY

unix

SYNTAX

```
#include <unix.h>
int creat(char *filename, int mode);
```

DESCRIPTION

`creat()` creates a new file named `filename`. That file is automatically opened in the specified mode. The mode argument is described in full under `open()`. The meaningful values for mode when using `creat()` are:

Opening flags:

<code>O_RDONLY</code>	read only
<code>O_WRONLY</code>	write only
<code>O_RDWR</code>	read/write
<code>O_APPEND</code>	append

Creation flags:

<code>O_CREAT</code>	create if file doesn't already exist
<code>O_TRUNC</code>	reset length to 0 (truncate) if file exists
<code>O_EXCL</code>	don't create if file exists

File type flags:

<code>O_TEXT</code>	text file
<code>O_BINARY</code>	binary file

OR a mode from "Opening Flags" to a mode from "File Type Flags" to create and open a specific file type in a specific way. If the file already exists (and `O_EXCL` is not set), `creat()` will delete and recreate it.

`creat()` returns a file number which is an index into `_file`, the array of file descriptors, and is used in calls to such functions as `read()`, `write()`, `close()`, and `lseek()`.

RETURN VALUE

The file number of the newly created and opened file if successful; EOF if error.

SEE ALSO

`open()`, `fopen()`, `FILE`, `_file`

- DISPLAY TIME

ctime

LIBRARY

unix

SYNTAX

```
#include <unix.h>
char *ctime (unsigned long *clock);
```

`ctime ()` returns a pointer to a string representing the time. The output is a 26 character string in the following form:

```
Sun Sep 16 01:03:52 1973\n\n0
```

If `clock` is `NULL`, the time is taken from the Macintosh's time buffer; otherwise the time passed in is used.

EXAMPLES

```
#include <unix.h>
unsigned long a;
time (&a); printf("%s", ctime(&a));
printf("%s", ctime(NULL));
```

RETURN VALUE

A pointer to a string containing the time.

SEE ALSO

`time()`, `localtime()`

•ERASE THE SCREEN

eraseplot

LIBRARY

unix

SYNTAX

```
#include <unix.h>
void eraseplot(void);
```

DESCRIPTION

`eraseplot()` erases (fills with white) the screen. The cursor is left at 0,0 (the upper left corner of the screen).

• LAUNCH A PROGRAM

execl

execv

execle

execve

LIBRARY

unix

SYNTAX

```
#include <stdio.h>
int execl(char *path);

int execv(char *path, char *argv[]);

int execle(char *path);

int execve(char *path, char *argv[], char *envp[]);
```

DESCRIPTION

These functions just launch the file name indicated by path. There is no support for arguments or environment variables. (Refer to the System V Unix manual for a complete explanation.)

RETURN VALUE

These functions never return.

- CLOSE FILES AND EXIT

exit

_exit

LIBRARY

unix

SYNTAX

```
#include <unix.h>
void exit(void);
void _exit(void);
```

DESCRIPTION

`exit()` calls `fclose()` for each open file and then calls `_exit()` to leave the program and return to the shell. Closing the files includes flushing the I/O buffers.

`_exit()` assumes that the files are closed and exits the program immediately.

`exit()` is used more often than `_exit()` because `exit()` takes care of pending I/O in the buffers while closing the files.

SEE ALSO

`abort()`, `fclose()`

- GET FILE NUMBER WHEN GIVEN FILE DESCRIPTOR

fileno

LIBRARY

unix

SYNTAX

```
#include <unix.h>
int fileno(FILE *stream);
```

DESCRIPTION

`fileno()` returns the file number of `stream`, where `stream` is a pointer to a file descriptor. The file number is simply the number of the file descriptor in the array of open file descriptors (see `_file` and `FILE`). Some library routines (`close()`, `read()`, `write()`, `lseek()`) access a file by using the file number, others by using the pointer to the actual file descriptor.

Files can be opened using either `fopen()` or `open()`; `fopen()` returns the pointer to the file descriptor, while `open()` returns the file number. Therefore, you can open a file with `fopen()` (returning the file descriptor pointer) and later use `fileno()` to get the file number and use the library functions which use the file number. There is no library function to go the other way. Instead use `_file[fn]` which is the file pointer of file number `fn`.

RETURN VALUE

The file number of `stream`, if success; EOF if error.

SEE ALSO

`_file`, `FILE`, `fopen()`, `open()`

- RETURN PROCESS ID NUMBER

getpid

LIBRARY

unix

SYNTAX

```
#include <unix.h>
int getpid(void);
```

DESCRIPTION

`getpid()` returns the process id number. This number is initially 1, but can be changed using `setpid()`.

This routine is provided for UNIX compatibility. It has no real function.

RETURN VALUE

The process id number.

SEE ALSO

`setpid()`

- RETURN USER ID NUMBER

getuid

LIBRARY

unix

SYNTAX

```
#include <unix.h>
int getuid(void);
```

DESCRIPTION

`getuid()` returns the user id number. This number is initially 1, but can be changed using `setuid()`.

This routine is provided for UNIX compatibility. It has no real function.

RETURN VALUE

The user's id number.

SEE ALSO

`setuid()`

- RETURN THE NEXT TWO BYTES OF STREAM AS INTEGER

getw

LIBRARY

unix

SYNTAX

```
#include <unix.h>
int getw(FILE *stream);
```

DESCRIPTION

getw() returns the next two bytes of stream as an integer. The bytes are assumed to be in memory image order (that is, according to the 68000 convention, with the MSB in the first byte).

EXAMPLE

```
int i, j;
i=0x1234;
.
.
.
fputc(i>>8 & 0x00ff, fp); /* MSB */
fputc(i & 0x00ff, fp);    /* lsb */
fseek(fp, -2L, 1);        /* move back 2 bytes */
j=getw(fp);               /* i==j */
```

RETURN VALUE

EOF is returned if an error occurs.

SEE ALSO

putw()

- GENERATE A SIGNAL

kill

LIBRARY

unix

SYNTAX

```
#include <stdio.h>
int kill(int pid, int sig);
```

DESCRIPTION

This generates a signal. Use getpid to determine the right value for pid.

RETURN VALUE

0 if successful, -1 otherwise (errno contains the actual error number).

- WRITE STRING AT CURSOR POSITION

label

LIBRARY

unix

SYNTAX

```
#include <unix.h>
void label(char *s);
```

DESCRIPTION

label() writes the null-terminated string s at the current cursor position.

The Macintosh coordinate system is defined with 0,0 at the top left of the screen, and 512,342 at the bottom right. Coordinates are given in pixels.

SEE ALSO

circle(), gotoxy(), line(), move(), point()

- PLOT LINE FROM X1,Y1 TO X2,Y2

line

LIBRARY

unix

SYNTAX

```
#include <unix.h>
void line(int x1, int y1, int x2, int y2);
```

DESCRIPTION

`line()` draws a line from pixel `(x1,y1)` to pixel `(x2,y2)` using Macintosh QuickDraw calls.

The Macintosh coordinate system is defined with 0,0 at the top left of the screen, and 512,342 at the bottom right. Coordinates are given in pixels.

SEE ALSO

`circle()`, `gotoxy()`, `label()`, `move()`, `point()`

- RETURN POINTER TO TIME RECORD

localtime

LIBRARY

unix

SYNTAX

```
#include <unix.h>
tm *localtime(unsigned long clock);
```

DESCRIPTION

localtime() returns a pointer to a UNIX-compatible time record called tm.

The record is of the following form (defined in unix.h):

```
typedef struct{

    int tm_sec;        /* seconds 0-59 */

    int tm_min;        /* minutes 0-59 */
    int tm_hour;       /* hours 0-23 */
    int tm_mday;       /* day of month (1-31) */
    int tm_mon;        /* month of year (0-11) */
    int tm_year;       /* year -1900 */
    int tm_wday;       /* day of week (sunday=0) */
    int tm_yday;       /* day of year (0-365) */
    int tm_isdst;      /* daylight savings time
                        - NOT SUPPORTED! */
} tm;
```

RETURN VALUE

A pointer to tm.

SEE ALSO

time(), ctime()

- LONG OUTPUT CONVERSION

locv

LIBRARY

unix

SYNTAX

```
#include <unix.h>
char *locv(int hi, int lo);
```

DESCRIPTION

`locv()` returns a signed double-precision integer (long), passed as two int arguments, to an ASCII string containing a decimal representation of that number. The value of the signed double-precision integer is $(hi \ll 16) \mid lo$.

This function is provided for UNIX compatibility.

RETURN VALUE

A pointer to a buffer containing the converted null terminated decimal string.

NOTE: Since `locv` returns a pointer to a static buffer it cannot be used more than once in an expression because the second call will overwrite the contents of the buffer.

- MOVE TO A DIFFERENT POINT INSIDE A FILE

lseek

LIBRARY

unix

SYNTAX

```
#include <unix.h>
long lseek(int fn, long offset, int type);
```

DESCRIPTION

`lseek()` allows for non-sequential I/O to a stream. When a file is opened for reading or writing, a "logical cursor" is placed at the beginning of the file. That logical cursor is marched forward through the file with each read or write. `lseek()` and its cousin `fseek()` move that logical cursor. `fseek()` is a standard C library function whereas `lseek()` is the UNIX version. (`fseek()` differs in two main ways: it takes a pointer to a file descriptor rather than a file number and it returns an `int` rather than a `long`.)

The `type` argument determines where the seek begins:

- 0 offset relative to the beginning of the file
- 1 offset relative to current file position
- 2 offset relative to the end of file

`offset` is simply the count in bytes from the starting point chosen by the `type` argument.

CAUTION: `offset` is a `long`, and can be either positive or negative.

RETURN VALUE

EOF if error; position in file if success

SEE ALSO

`fseek()`

- MOVE CURSOR TO X,Y

move

LIBRARY

unix

SYNTAX

```
#include <unix.h>
void move(int x, int y);
```

DESCRIPTION

move () moves the cursor to Macintosh pixel position x, y.

The Macintosh coordinate system is defined with 0,0 at the top left of the screen, and 512,342 at the bottom right. Coordinates are given in pixels.

SEE ALSO

circle (), gotoxy (), label (), line (), point ()

- MOVE N BYTES FROM ONE POSITION TO ANOTHER

movmem

LIBRARY

unix

SYNTAX

```
#include <unix.h>
void movmem(char *s, char *d, unsigned n);
```

DESCRIPTION

movmem() copies n bytes from location s in memory to location d. This is a block copy operation. The routine checks the relative positions of the source and destination to ensure that data is not overwritten during the move.

EXAMPLES

```
char s[100];
char t[200];
movmem(&t[100], s, sizeof(s));
/* copies last 100 elements of t into s */
```

SEE ALSO

repmem(), setmem()

• OPEN FILENAME

open

LIBRARY

unix

SYNTAX

```
#include <unix.h>
int open(char *filename, int mode);
```

DESCRIPTION

`open()` opens a filename for reading or writing, according to mode. `open()` returns the file number of the open file. That file number is used by the library functions `read()`, `write()`, `lseek()`, and `close()`. mode is made by adding together one flag from each of the three groups below.

Opening flags:

<code>O_RDONLY</code>	read only
<code>O_WRONLY</code>	write only
<code>O_RDWR</code>	read/write
<code>O_APPEND</code>	append

Creation flags:

<code>O_CREAT</code>	create if file doesn't already exist
<code>O_TRUNC</code>	reset length to 0 (truncate) if file exists
<code>O_EXCL</code>	don't create if file exists

File type flags:

<code>O_TEXT</code>	text file
<code>O_BINARY</code>	binary file

`fopen()` also opens a file, but uses different arguments and returns a file pointer rather than a file number.

`creat()` uses the opening flags and file type flags described here when creating a file. The creation flags are defaulted by `creat()` to `O_CREAT` and `O_TRUNC`.

RETURN VALUE

`n`, where `n` is the file number of the file (0-14), if success; EOF if error.

SEE ALSO

`close()`, `creat()`, `read()`, `write()`, `lseek()`, `fopen()`

- PRINT USER'S STRING AND ERROR NUMBER

perror

LIBRARY

unix

SYNTAX

```
#include <unix.h>
void perror(char *str);
```

DESCRIPTION

perror() prints out the user's string (str) on stdout and then "Error number n" where n is the last error number that occurred.

RETURN VALUE

void

SEE ALSO

exit()

- PLACE A POINT AT LOCATION X,Y

point

LIBRARY

unix

SYNTAX

```
#include <unix.h>
void point(int x, int y);
```

DESCRIPTION

point (x,y) sets pixel (x,y) on the Macintosh screen to black. The Macintosh coordinate system is defined with 0,0 at the top left of the screen, and 512,342 at the bottom right.

Note: do not confuse this function with the QuickDraw type Point .

SEE ALSO

circle(), gotoxy(), label(), line(), move()

- WRITE WORD AS 2 BYTE NUMBER TO STREAM

putw

LIBRARY

unix

SYNTAX

```
#include <unix.h>
int putw(int word, FILE *stream);
```

DESCRIPTION

`putw()` writes word as high byte, then low byte to the given output stream. Words written to a file are in memory image format.

RETURN VALUE

The word if success; EOF if error.

SEE ALSO

`getw()`

- SORT A TABLE IN PLACE

qksort

LIBRARY

unix

SYNTAX

```
#include <unix.h>
void qksort(int nel, int (*compare)(),
            int (*swap)());
```

DESCRIPTION

qksort is an implementation of the quicksort algorithm. It sorts a table of data.

nel is the number of elements in the table.

compare is the name of a user-supplied comparison function.

swap is the name of a user-supplied swap function.

```
int compare(int i, int j);
```

The compare function is passed two indices, starting at zero, into the table, the location of which is assumed to be known by compare. compare should return an integer less than, equal to, or greater than zero, depending on whether the first indexed item is less than, equal to, or greater than the second.

```
void swap(int i, int j);
```

The swap function is passed two indices, starting at zero, into the table, the location of which is assumed to be known by swap. swap should interchange the two table items.

SEE ALSO

qsort()

- SORT A TABLE IN PLACE

qsort

LIBRARY

unix

SYNTAX

```
#include <unix.h>
int qsort(char *base, int nel, int size,
          int (*compare)());
```

DESCRIPTION

qsort is an implementation of the quicksort algorithm. It sorts a table of data.

base points to the first byte of the table.

nel is the number of elements in the table.

size is the size of each element in the table. NOTE: to provide an efficient implementation for the internal swap procedure, it is assumed that if size is 2 or 4, then the table pointed to by base is aligned on a word boundary. If it is not, qsort will cause an *address error*.

compare is the name of a user-supplied comparison function.

```
int compare(type *p1, type *p2);
```

The compare function is passed two pointers to items of *type*. compare should return an integer less than, equal to, or greater than zero, depending on whether the first argument is less than, equal to, or greater than the second.

SEE ALSO

qksort()

• READ CHARACTERS FROM A FILE

read

LIBRARY

unix

SYNTAX

```
#include <unix.h>
int read(int fn, char *buffer, unsigned int count);
```

DESCRIPTION

`read()` tries to read `count` bytes from file number `fn` into the array pointed to by `buffer`. If `read()` reaches the EOF before it has read `count` bytes, it stops and returns the number of bytes it did read. Make sure the buffer is large enough to hold the data. An efficient size for reading is `BUFSIZ`, the size of the I/O buffers. `fread()` also reads data in from a file but uses different arguments, including accessing the file by its file pointer.

RETURN VALUE

The number of bytes read; EOF if error.

SEE ALSO

`fread()`, `write()`, `open()`

- REMOVE FILENAME FROM DIRECTORY

remove

LIBRARY

unix

SYNTAX

```
#include <unix.h>
int remove(char *filename);
```

DESCRIPTION

`remove()` deletes the file from the directory in its entirety. It performs the same function as `unlink` in this implementation.

RETURN VALUE

0 if success; -1 if failure.

SEE ALSO

`unlink()`, `rename()`, `creat()`

- RENAME A FILE

rename

LIBRARY

unix

SYNTAX

```
#include <unix.h>
int rename(char *old, char *new);
```

DESCRIPTION

`rename()` changes the name of a file in the directory from `old` to `new`. `old` and `new` are C-style null terminated strings.

RETURN VALUE

0 if success; -1 if failure.

SEE ALSO

`creat()`, `remove()`, `rename()`

• COPY PATTERN INTO MEMORY

repmem**LIBRARY**

unix

SYNTAX

```
#include <unix.h>
void repmem(char *s, char *v, int lv, int nv);
```

DESCRIPTION

repmem copies the pattern pointed to by *v* into memory at *s*, *nv* times. *lv* is the length of the pattern string *v*.

This means that $nv * lv$ bytes are moved into *v*.

SEE ALSO

`allmem()`, `bldmem()`, `getmem()`, `movmem()`, `rlsmem()`,
`rstmem()`, `setmem()`

- ATTACH PRIVATE BUFFER TO FILE

setbuf

LIBRARY

unix

SYNTAX

```
#include <unix.h>
void setbuf(FILE *fp, char *buf);
```

DESCRIPTION

setbuf() attaches a private buffer to the file whose file pointer is fp. The size of the buffer is BUFSIZ. If buf is NULL then this is the same as calling setnbf.

SEE ALSO

setnbuf()

- PERFORM NON-LOCAL GOTO

setjmp

longjmp

LIBRARY

unix

SYNTAX

```
#include <unix.h>
int setjmp(jmp_buf env);

void longjmp(jmp_buf env, int val);
```

DESCRIPTION

`setjmp()` is used with `longjmp()` to provide a non-local goto function, useful for dealing with errors and interrupts in low-level subroutines.

`setjmp()` saves the stack environment in `env` and returns the value 0. `longjmp` restores the environment saved by the last call to `setjmp()`. Program execution continues as if `setjmp` had just returned `val`.

RETURN VALUE

`setjmp()` returns 0 unless `longjmp()` is called with `val=0`, in which case `setjmp()` returns 1.

- SET A FILE TO BE UNBUFFERED

setnbuf

LIBRARY

unix

SYNTAX

```
#include <unix.h>
void setnbuf(FILE *fp);
```

DESCRIPTION

`setnbuf` sets a file as unbuffered; that is, it undoes the effect of a previous call to `setbuf()`.

SEE ALSO

`setbuf()`

- SET PROCESS ID

setpid

LIBRARY

unix

SYNTAX

```
#include <unix.h>
int setpid(int val);
```

DESCRIPTION

setpid() sets the process id number to val.

This call is provided solely for UNIX compatibility. It has no real function.

RETURN VALUE

val, the argument passed in.

SEE ALSO

getpid()

- SET USER'S ID NUMBER TO VAL

setuid

LIBRARY

unix

SYNTAX

```
#include <unix.h>
int setuid(int val);
```

DESCRIPTION

setuid sets the user's id number to val.

This routine is provided for UNIX compatibility. It has no real function.

RETURN VALUE

val, the argument passed in.

SEE ALSO

getuid()

• SET A SIGNAL

signal**LIBRARY**

unix

SYNTAX

```
#include <stdio.h>
int (*signal(int sig, int (*func)()))()
```

DESCRIPTION

Setting the SIGINT signal allows C and . (command-period) to be trapped. Setting the SIGTERM signal is just like calling onexit. Other signals can be set, but are only generated by explicit calls to kill. Each signal is automatically reset to SIG_DFL once invoked.

Signals set to SIG_IGN are ignored. All signals are initially set to SIG_IGN. Signals set to SIG_DFL will cause the program to exit, or will break to the debugger if one is present.

RETURN VALUE

The previous function set for sig.

• PAUSE SECONDS, THEN CONTINUE

sleep

LIBRARY

unix

SYNTAX

```
#include <unistd.h>
void sleep(int secs);
```

DESCRIPTION

sleep() waits secs and then continues.

- CONVERT AN ASCII INTEGER VALUE TO AN INTEGER

stcd_i

LIBRARY

unix

SYNTAX

```
#include <unix.h>
int stcd_i(char *s, int *r);
```

DESCRIPTION

`stcd_i()` converts an ASCII string `s` to an integer. The result is placed in `r` and the number of characters scanned is returned. Scanning stops when an invalid character is present. Valid characters are 0-9, -, and +.

`stcd_i()` is similar to `atoi()`.

RETURN VALUE

The number of characters scanned.

SEE ALSO

`stch_i()`, `stci_d()`, `stcu_d()`, `atoi()`

• CONVERT AN ASCII HEX VALUE TO AN INTEGER

stch_i

LIBRARY

unix

SYNTAX

```
#include <unix.h>
int stch_i(char *s, int *r);
```

DESCRIPTION

`stch_i()` converts an ASCII hex string `s` to an integer. The result is placed in `r` and the number of characters scanned is returned. Scanning stops when an invalid character is present. Valid characters are 0-9, a-f, A-F, -, and +.

RETURN VALUE

The number of characters scanned.

SEE ALSO

`stcd_i()`, `stci_d()`, `stcu_d()`

• CONVERT INTEGER TO ASCII STRING, PLACE STRING IN BUFFER

stci_d

LIBRARY

unix

SYNTAX

```
#include <unix.h>
int stci_d(char *out, int in, int outlen);
```

DESCRIPTION

`stci_d()` converts the integer `in` to an ASCII string and places the string in the buffer pointed to by `out`. A maximum of `outlen` characters are placed into the buffer.

RETURN VALUE

Number of characters placed in buffer `out`, not including null terminator.

SEE ALSO

`stch_i()`, `std_i()` `stcu_d()`

• CONVERT INTEGER TO ASCII STRING, PLACE STRING IN BUFFER

stcu_d

LIBRARY

unix

SYNTAX

```
#include <unix.h>
int stcu_d(char *out, unsigned in, int outlen);
```

DESCRIPTION

`stcu_d()` converts the unsigned integer `in` to an ASCII string and places the string in the buffer pointed to by `out`. A maximum of `outlen` characters are placed into the buffer.

RETURN VALUE

The number of characters placed in buffer `out`, not including null terminator.

SEE ALSO

`stcd_i()`, `stch_i()`, `stci_d()`

• PARSE FILENAME PATTERN

stspfp**LIBRARY**

unix

SYNTAX

```
#include <unix.h>
int stspfp(char *p, int n[16]);
```

DESCRIPTION

stspfp parses a filename pattern which consists of node names separated by colons. Each colon is replaced by a null byte and the beginning index of that node is placed in the index array.

For example, :abc:de: fgh has 3 nodes, and their indices are 1 for abc, 5 for de and 8 for fgh. The last entry in the array index is -1.

RETURN VALUE

0 if successful; -1 if too many nodes or other error.

• RETURN POSITION WITHIN FILE

tell

LIBRARY

unix

SYNTAX

```
#include <unix.h>
long tell(int fildes);
```

DESCRIPTION

`tell` returns the position within the file. This is the UNIX equivalent to `ftell`, except you pass the file's number instead of the file descriptor. If an error occurs then EOF is returned.

RETURN VALUE

A long representing the position within file if success; EOF if error.

SEE ALSO

`ftell()`

- RETURN THE TIME IN SECONDS

time

LIBRARY

unix

SYNTAX

```
#include <unix.h>
unsigned long time(unsigned long *tloc);
```

DESCRIPTION

`time()` returns the time in seconds since Jan 1, 1970, 00:00:00. If `tloc` is not `NULL`, then the time is placed in `tloc` also.

RETURN VALUE

The time in seconds since Jan 1, 1970, 00:00:00.

SEE ALSO

`ctime()`, `localtime()`

• RETURN TERMINAL ID NUMBER

ttyn

LIBRARY

unix

SYNTAX

```
#include <unix.h>
int ttyn(void)
```

DESCRIPTION

ttyn() returns the terminal id number, which is 1. This routine is provided for UNIX compatibility, and performs no real function.

RETURN VALUE

The terminal id number, which is 1.

SEE ALSO

setpid(), setuid(), getuid(), getpid()

- UNLINK A FILE FROM THE DIRECTORY

unlink

LIBRARY

unix

SYNTAX

```
#include <unix.h>
int unlink(char *filename);
```

DESCRIPTION

unlink() removes filename from the directory and deletes the file. The library function remove() also removes a file from the directory.

RETURN VALUE

0 if success; -1 if failure.

SEE ALSO

creat(), remove(), rename()

- WRITE A BLOCK OF BYTES TO A FILE

write

LIBRARY

unix

SYNTAX

```
#include <unix.h>
int write(int fn, char *buffer, unsigned nbytes);
```

DESCRIPTION

`write()` copies `nbytes` from `buffer` to the file with file descriptor number `fn`. The most efficient choice for `nbytes` is usually `BUFSIZ`.

`fwrite()` is another library function to write a block of data to a file, with some differences. `write()` uses the file number (returned if `open()` is used to open the file) while `fwrite()` uses the file descriptor pointer (returned if `fopen()` is used to open the file). In addition, `write()` has three arguments, while `fwrite()` has four. Therefore, `write()` and `fwrite()` are NOT synonymous.

RETURN VALUE

The number of bytes copied, if success; -1 if error.

SEE ALSO

`fwrite()`



unix_strings 7

Introduction

The `unix_strings` library provides some string functions found in UNIX environments. Some of the routines here have the same functionality as routines in the `strings` library. They are included for compatibility..

- COUNT NUMBER OF CHARACTERS IN STRING THAT ARE IN SET

stcarg

LIBRARY

unix_strings

SYNTAX

```
#include <strings.h>
int stcarg(char *s, char *set);
```

DESCRIPTION

stcarg() counts the number of characters in string s that are in set.

stcarg() will ignore strings enclosed in double quotes or literals enclosed in quotes inside s (see example). A backslash character in s is also ignored.

Consequently, a single quote, a double quote, or a backslash contained in *set will never match a character in s.

stcarg() returns the count of matched characters.

EXAMPLES

```
stcarg("abcde","12345") /* returns 0 */
stcarg("aabcd","aeiou") /* returns 2 */
stcarg("aeiou","aabcd") /* returns 1 */
stcarg("abc\\d","abc\\d")
/* returns 4 (the backslash character is ignored) */

stcarg("'s'tp","stp")
/* returns 2 (the s is ignored because it is quoted)
*/
```

RETURN VALUE

The number of characters in s that are in set with the restrictions described above.

- SEARCH FOR PATTERN ANYWHERE IN STRING

stcpm

LIBRARY

unix_strings

SYNTAX

```
#include <strings.h>
int stcpm(char *string, char *pattern, char *q);
```

DESCRIPTION

stcpm() calls stcpma() repetitively to search for pattern anywhere in string.

stcpm() allows wildcard matches in pattern, using the following pattern-matching rules:

- ? matches any single character.
- c* (where c can be replaced by any character) matches any number of consecutive c characters (including 0).
- c+ (where c can be replaced by any character) matches one or more consecutive c characters (but not 0 characters).

To match a wildcard character literally, precede it by a backslash (\) character. For example, * in pattern will only match a single * in string, \? will only match a ?, and \+ will only match a + (When you are writing the backslash character in a string for the compiler, you have to precede the backslash character itself with a backslash. \\ is turned by the compiler into a single backslash character. In input from the keyboard or a file, a single backslash is sufficient.).

Because of wildcards and backslashes, the number of characters matched is not necessarily the same in pattern as in string (for example, pattern = "ab+cd", string = "abbbbbcd" is a perfect match even though pattern has 5 characters and string has 8).

NOTE: stcpm() returns two values, one directly and one by side effect. Its return value is an int which is the length of the matched pattern if a match is made, and 0 if no match is made.

*q, the third argument, is used to return a pointer to the first occurrence of pattern in string (or is left unchanged if no match is made).

EXAMPLES

```
char *substring;  
stcpm("bxde", "a*b+?d", &substring)  
  
/* returns 3 and sets substring to the address of  
the start of the source string Note that a* matches  
zero or more  
instances of "a".*/
```

```
stcpm ("yybxde", "a*b+?d", &substring)
```

```
/* returns 3 and sets substring to the address of  
'b' in the source string.*/
```

RETURN VALUE

0 if pattern is not in string; n where n is the length of pattern if pattern is in string; by side effect, the third argument *q is set to p where p is a pointer to the first pattern in string. *q is left unchanged if no match occurs.

SEE ALSO

```
stcpma ()
```

• MATCH PATTERN AT START OF STRING

stcpma**LIBRARY**

unix_strings

SYNTAX

```
#include <strings.h>
int stcpma(char *string, char *pattern);
```

DESCRIPTION

stcpma() compares pattern to the beginning of string. If they match exactly, stcpma() returns the number of characters in the match. If they partially match, stcpma() returns the number of characters until they differ. If they don't match at all, stcpma() returns 0.

stcpma() supports the following wildcard matching syntax in pattern:

- ? matches any single character.
- c* (where c can be replaced by any character) matches any number of consecutive c characters (including 0).
- c+ (where c can be replaced by any character) matches one or more consecutive c characters (but not 0 characters).

To match a wildcard character literally, precede it by a backslash (\) character. For example, * in pattern will only match a single * in string, \? will only match a ?, and \+ will only match a + (When you are writing the backslash character in a string for the compiler, you have to precede the backslash character itself with a backslash. \\ is turned by the compiler into a single backslash character. In input from the keyboard or a file, a single backslash is sufficient.).

Because of wildcards and backslashes, the number of characters matched is not necessarily the same in pattern as in string (for example, pattern = "ab+cd", string = "abbbbbcd" is a perfect match even though pattern has 5 characters and string has 8). The number returned is the number of characters in string that were matched.

EXAMPLES

```
stcpma("bxde", "a*b+?d");  
/* returns 3 */
```

```
stcpma("yybxde", "a*b+?d");  
/* returns 0 */
```

RETURN VALUE

The number of characters at the beginning of `string` that match `.br` pattern; 0 if no characters match.

SEE ALSO

`stcpm()`

- PARSE FIRST SYMBOL IN INPUT

stpsym

LIBRARY

unix_strings

SYNTAX

```
#include <strings.h>
char *stpsym(char *input, char *output, int symlen);
```

DESCRIPTION

`stpsym()` finds the first symbol in `input`. A symbol begins with a letter (upper or lower case) and ends at the first character which is not a letter or digit. `stpsym()` copies up to the first (`symlen - 1`) characters of the first symbol from `input` to `output`.

The `symlen` argument should be the length of output available, as it keeps `stpsym()` from copying too much into output and destroying the memory that follows. A null character (`'\0'`) is added to output after the last character from the symbol.

`stpsym()` also returns a pointer to the first character in `input` after the symbol. If the first character in `input` does not begin a symbol, `stpsym()` does not copy into output and returns a pointer to the first character in `input`.

Note that if the first symbol in `input` is larger than (`strlen - 1`) characters, `stpsym()` returns a pointer to the next character in that symbol as the beginning of the next symbol. Thus, if the character at the return pointer is a letter or digit, you know that the preceding symbol has been broken in two.

The key thing to remember is that `stpsym()` returns two things: a copy of the symbol in `output` and a pointer to the character after the symbol in `input`.

EXAMPLES

```
stpsym(" abc_1 ", sym, sizeof(sym))
/* sets sym = "abc" */
```

RETURN VALUE

Directly: `p` where `p` is a pointer into `string` to the character after the symbol. `p` points to the beginning of `s` if it does not begin a symbol.

By side effect, output has a copy of the first symbol from `string` (up to `(strlen - 1)` characters) terminated with a `'\0'`.

SEE ALSO

`stptok()`

• PARSE FIRST TOKEN IN INPUT

stptok**LIBRARY**

unix_strings

SYNTAX

```
#include <strings.h>
char *stptok(char *input, char *output, int toklen,
char *brkset);
```

DESCRIPTION

`stptok()` parses the next token in `input`. A token is defined as a sequence of characters not containing any characters from `brkset`, the fourth argument to `stptok()`.

`stptok()` has two effects. First, `stptok()` copies up to the first `(toklen - 1)` characters from the first token in `input` to `output` and appends a null character (`'\0'`) to `output`. Second, `stptok()` returns a pointer to the first character after the token.

There are two special cases to be aware of. If the first character in `input` is not part of a token (and therefore a member of the `brkset`), `stptok()` puts a `'\0'` at the start of `output` and returns a pointer to that first character in `input`. If the first token in `input` is longer than `(toklen - 1)` characters, the pointer returned is `&input[toklen]`, even though that character might seem to be logically part of the first token.

`stptok()` is parallel to `stpsym()` in every way. `stpsym()` parses the first symbol (letters and digits) of an input stream.

EXAMPLES

```
stptok("abc to 2", sym, sizeof (sym), "5")
```

```
/* returns a pointer to the first character
   in the source and sets sym to "abc" */
```

RETURN VALUE

A pointer to the next character in `input` after the first token is parsed; by side effect, `output` contains a copy of the first `(toklen - 1)` characters of the first token in `input`. `output` is always terminated with a `'\0'`.

SEE ALSO`stpsym()`

Standard Libraries Reference



storageu 8

Introduction

The `storageu` library provides low level memory management found in Unix environments. These routines call the Macintosh memory manager to build a storage pool.

Standard Libraries Reference

• ALLOCATE LEVEL 2 MEMORY POOL

allmem

LIBRARY

storageu

SYNTAX

```
#include <storage.h>
int allmem(void);
```

DESCRIPTION

`allmem()` is equivalent to the call: `bldmem(0)`.

Note: The UNIX library procedures `getmem()`, `getml()`, `rlsmem()`, `rlsml()`, `allmem()`, `bldmem()`, `rbrk()`, `rstmem()`, `lsbrk()`, and `sbrk()` are not recommended for casual use. These procedures attempt to provide low-overhead UNIX-like memory allocation, and are provided for compatibility purposes.

RETURN VALUE

0 if success; -1 if failure.

SEE ALSO

`bldmem()`, `rbrk()`, `rstmem()`

- SET UP LEVEL 2 MEMORY POOL

bldmem

LIBRARY

storageu

SYNTAX

```
#include <storage.h>
int bldmem(int n)
```

DESCRIPTION

`bldmem()` allocates a pool of `n` K bytes of memory for subsequent `getmem()` and `getml()` calls.

Typically, you would make some initial calls to `lsbrk()` or `sbrk()` to get "level 1" memory. Then you would make a call to `allmem()` or `bldmem()` to set up a level 2 memory pool.

Memory allocated by `getmem()` and `getml()` calls after a call to `allmem()` or `bldmem()` can all be released by a call to `rstmem()`.

Note: The UNIX library procedures `getmem()`, `getml()`, `rlsmem()`, `rlsml()`, `allmem()`, `bldmem()`, `rbrk()`, `rstmem()`, `lsbrk()`, and `sbrk()` are not recommended for casual use. These procedures attempt to provide low-overhead UNIX-like memory allocation, and are provided for compatibility purposes.

RETURN VALUE

0 if success; -1 if failure.

SEE ALSO

`allmem()`, `rbrk()`, `rstmem()`

Standard Libraries Reference

- DYNAMICALLY ALLOCATE N BYTES OF MEMORY

getmem

LIBRARY

storageu

SYNTAX

```
#include <storage.h>
char *getmem(unsigned n)
```

DESCRIPTION

getmem() gets n bytes of memory from the free memory pool allocated by bldmem(). The memory is not automatically zeroed.

getml() is the same as getmem() except that it can allocate a block of memory that can be greater than 65535 bytes. There are actually a half dozen different functions which dynamically allocate memory. They are described together in the introduction to this section as well as on the separate pages describing each function.

Use rlsmem() to free memory allocated by getmem().

Note: The UNIX library procedures getmem(), getml(), rlsmem(), rlsml(), allmem(), bldmem(), rbrk(), rstmem(), lsbrk(), and sbrk() are not recommended for casual use. These procedures attempt to provide low-overhead UNIX-like memory allocation, and are provided for compatibility purposes.

RETURN VALUE

A pointer to the newly allocated block; or NULL if error (such as insufficient memory).

SEE ALSO

getml(), rlsmem(), rlsml()

- DYNAMICALLY ALLOCATE N BYTES OF MEMORY

getml

LIBRARY

storageu

SYNTAX

```
#include <storage.h>
char *getml(unsigned long n);
```

DESCRIPTION

getml () gets n bytes of memory from the free memory pool allocated by bldmem(). The memory is not automatically zeroed. getmem () is the same as getml () except that it can allocate only an integer length block of memory.

There are actually a half dozen different functions which dynamically allocate memory. They are described together in the introduction to this section as well as separately by name.

Use rlsml () to free memory allocated by getml ().

Note: The UNIX library procedures getmem(), getml(), rlsmem(), rlsml(), allmem(), bldmem(), rbrk(), rstmem(), lsbrk(), and sbrk() are not recommended for casual use. These procedures attempt to provide low-overhead UNIX-like memory allocation, and are provided for compatibility purposes.

RETURN VALUE

A pointer to the newly allocated block; or NULL if error (such as insufficient memory).

SEE ALSO

getmem(), rlsml()

- DYNAMICALLY ALLOCATE A BLOCK OF N BYTES

lsbrk

LIBRARY

storageu

SYNTAX

```
#include <storage.h>
char *lsbrk(unsigned long n)
```

DESCRIPTION

lsbrk () allocates a long block of bytes. The block is not zeroed. This is a low-level UNIX call for getting memory. There are half a dozen different ways to allocate memory. They are described together in the introduction to this section as well as individually by function name.

Note: The UNIX library procedures getmem(), getml(), rlsmem(), rlsml(), allmem(), bldmem(), rbrk(), rstmem(), lsbrk(), and sbrk() are not recommended for casual use. These procedures attempt to provide low-overhead UNIX-like memory allocation, and are provided for compatibility purposes.

RETURN VALUE

NULL if error; the pointer to the newly allocated block if success.

SEE ALSO

rstmem(), sbrk(), getml(), getmem()

- RESET MEMORY BREAK POINT

rbrk

LIBRARY

storageu

SYNTAX

```
#include <storage.h>
void rbrk(void);
```

DESCRIPTION

rbrk() resets the memory break point to the starting position.

Note: The UNIX library procedures getmem(), getml(), rlsmem(), rlsml(), allmem(), bldmem(), rbrk(), rstmem(), lsbrk(), and sbrk() are not recommended for casual use. These procedures attempt to provide low-overhead UNIX-like memory allocation, and are provided for compatibility purposes.

SEE ALSO

allmem(), bldmem(), rstmem(), lsbrk(), sbrk(), getmem(), rlsmem(), getml(), rlsml(), rstmem()

- CHANGE THE SIZE OF AN ALLOCATED REGION OF MEMORY

realloc

LIBRARY

storageu

SYNTAX

```
#include <storage.h>
char *realloc(char *ptr, unsigned int size)
```

DESCRIPTION

`realloc()` changes the size of the memory region pointed to by `ptr` while preserving its contents.

If the new size is larger than the old, new space is added at the end. If necessary, the contents are copied to another region of memory. If the new size is smaller than the old, space is taken away at the end of the old region and the contents of that portion are lost.

`relalloc()` can be used if you need a larger size specification (long rather than int).

RETURN VALUE

A pointer to the (new) region in memory if success; NULL if error.

SEE ALSO

`relalloc()`

- CHANGE THE SIZE OF AN ALLOCATED REGION OF MEMORY

relalloc

LIBRARY

storageu

SYNTAX

```
#include <storage.h>
char *relalloc(char *ptr, unsigned lonsize)
```

DESCRIPTION

`relalloc` changes the size of the memory region pointed to by `ptr` while preserving its contents. If necessary, the contents are copied to another region of memory.

This function is identical to `realloc` except that it allows for a larger size specification (long rather than int).

RETURN VALUE

A pointer to the (new) region in memory if success; NULL if error.

SEE ALSO

`realloc()`

- RELEASE MEMORY ALLOCATED BY GETMEM()

rlsmem

LIBRARY

storageu

SYNTAX

```
#include <storage.h>
int rlsmem(char *cp, unsigned n)
```

DESCRIPTION

`rlsmem()` releases a block of memory allocated by `getmem()`. `cp` is a pointer to the first character in the block, and `n` is an unsigned length of the block.

`rlsmem()` is analogous to `free()`, which releases a block of memory allocated by `calloc()`. Note the difference between `rlsmem()` and `free()`: `rlsmem()` needs to be told how many bytes to release, while `free()` releases the whole block automatically. For that reason, `rlsmem()` is both more flexible (can release part of a block) and more difficult to use than `free()`.

`rlsml()` releases memory allocated by `getml()` or `getmem()`.

The introduction to this section contains a list matching allocating functions with deallocating functions.

Note: The UNIX library procedures `getmem()`, `getml()`, `rlsmem()`, `rlsml()`, `allmem()`, `bldmem()`, `rbrk()`, `rstmem()`, `lsbrk()`, and `sbrk()` are not recommended for casual use. These procedures attempt to provide low-overhead UNIX-like memory allocation, and are provided for compatibility purposes.

RETURN VALUE

0 if success; -1 if failure.

SEE ALSO

`getmem()`, `rlsml()`, `getml()`

- RELEASE MEMORY GOT BY GETML()

rlsml

LIBRARY

storageu

SYNTAX

```
#include <storage.h>
int rlsml(char *cp, unsigned long n);
```

DESCRIPTION

`rlsml()` releases a block of memory got by `getml()`. `cp` is a pointer to the first character in the block, and `n` is an unsigned length of the block.

`rlsml()` is analogous to `free()`, which releases a block of memory allocated by `calloc()`. Note the difference between `rlsml()` and `free()`: `rlsml()` needs to be told how many bytes to release, while `free()` releases the whole block automatically. For that reason, `rlsml()` is both more flexible (can release part of a block) and more difficult to use than `free()`.

`rlsmem()` releases memory allocated by `getmem()`.

The introduction to this section contains a list matching allocating functions with deallocating functions.

Note: The UNIX library procedures `getmem()`, `getml()`, `rlsmem()`, `rlsml()`, `allmem()`, `bldmem()`, `rbrk()`, `rstmem()`, `lsbrk()`, and `sbrk()` are not recommended for casual use. These procedures attempt to provide low-overhead UNIX-like memory allocation, and are provided for compatibility purposes.

RETURN VALUE

0 if success; -1 if failure.

SEE ALSO

`getml()`, `rlsmem()`

- RESET MEMORY ALLOCATED BY GETMEM()

rstmem

LIBRARY

storageu

SYNTAX

```
#include <storage.h>
void rstmem(void);
```

DESCRIPTION

`rstmem()` releases the memory allocated by `getmem()` calls made after calls to `allmem()` or `bldmem()`.

Note: The UNIX library procedures `getmem()`, `getml()`, `rlsmem()`, `rlsml()`, `allmem()`, `bldmem()`, `rbrk()`, `rstmem()`, `lsbrk()`, and `sbrk()` are not recommended for casual use. These procedures attempt to provide low-overhead UNIX-like memory allocation, and are provided for compatibility purposes.

SEE ALSO

`allmem()`, `bldmem()`, `rbrk()`, `sbrk()`, `lsbrk()`, `br getmem()`, `getml()`, `rlsmem()`, `rlsml()`

- DYNAMICALLY ALLOCATE A BLOCK OF N BYTES

sbrk

LIBRARY

storageu

SYNTAX

```
#include <storage.h>
char *sbrk(unsigned n)
```

DESCRIPTION

sbrk() dynamically allocates a block of n bytes. The block is not zeroed. A similar function, calloc(), allocates a block and zeroes it before returning.

rbrk() is used to release memory allocated by sbrk(), lsbrk(), getmem() or getml().

Note: The UNIX library procedures getmem(), getml(), rlsmem(), rlsml(), allmem(), bldmem(), rbrk(), rstmem(), lsbrk(), and sbrk() are not recommended for casual use. These procedures attempt to provide low-overhead UNIX-like memory allocation, and are provided for compatibility purposes.

RETURN VALUE

A pointer to the newly allocated block; NULL if error.

SEE ALSO

lsbrk(), getmem(), getml()

- INITIALIZE A BLOCK OF MEMORY

setmem

LIBRARY

storageu

SYNTAX

```
#include <unix.h>
void setmem(char *addr, unsigned n; char c);
```

DESCRIPTION

setmem() initializes a block of memory starting at addr, for n bytes, to the value c.

SEE ALSO

movmem(), repmem()

Index

_closeall..... 9
 _exit..... 147
 _tolower..... 67
 _toupper..... 68
 abort..... 136
 abs..... 101
 acos..... 102
 allmem..... 198
 asin..... 103
 atan..... 104
 atan2..... 105
 atof..... 137
 atoi..... 137
 atol..... 137
 bldmem..... 199
 calloc..... 128
 ceil..... 106
 cfree..... 129
 cgetpid..... 138
 cgets..... 6
 circle..... 139
 clalloc..... 130
 clearerr..... 7
 Click_on..... 8
 close..... 140
 clrerr..... 7
 cont..... 141
 cos..... 107
 cosh..... 108
 cprintf..... 10
 cputs..... 11
 creat..... 142
 cscanf..... 12
 ctime..... 144
 ctype..... 13
 Echo_to_Printer..... 15
 eraseplot..... 145
 execl..... 146
 execl..... 146
 execv..... 146
 execve..... 146
 exit..... 147
 exp..... 109
 fabs..... 110
 fclose..... 16
 feof..... 17
 ferror..... 18

fflush..... 19
 fgetc..... 20
 fgets..... 21
 fileno..... 148
 floor..... 111
 fmod..... 112
 fopen..... 22
 fopenw..... 24
 fprintf..... 25
 fputc..... 26
 fputs..... 27
 fread..... 28
 free..... 131
 freopen..... 29
 frexp..... 113
 fscanf..... 30
 fseek..... 31
 ftell..... 32
 fwrite..... 33
 Get_ScreenPtr..... 40
 getc..... 34
 getch..... 35
 getchar..... 36
 getche..... 37
 getmem..... 200
 getml..... 201
 getpid..... 149
 gets..... 38
 getuid..... 150
 getw..... 151
 getxpos..... 39
 getypos..... 39
 gotoxy..... 41
 kbhit..... 42
 kill..... 152
 label..... 153
 labs..... 114
 ldexp..... 115
 line..... 154
 localtime..... 155
 locv..... 156
 log..... 116
 log10..... 117
 longjmp..... 171
 lsrk..... 202
 lseek..... 157
 malloc..... 132

Standard Libraries Reference

malloc.....	133	stci_d.....	179
modf.....	118	stcis.....	75
move.....	158	stciscn.....	76
movmem.....	159	stclen.....	77
onexit.....	43	stcpm.....	189
open.....	160	stcpma.....	191
perror.....	161	stcu_d.....	180
point.....	162	std_ver.....	65
pow.....	119	StdEvent.....	62
printf.....	44	Stdio_config.....	63
putc.....	48	Stdio_MacInit.....	64
putch.....	49	stpblk.....	78
putchar.....	50	stpbrk.....	79
puts.....	51	stpchr.....	80
putw.....	163	stpcpy.....	81
qksort.....	164	stpsym.....	193
qsort.....	165	stptok.....	195
rand.....	120	strcat.....	82
rbrk.....	203	strchr.....	83
read.....	166	strcmp.....	84
realloc.....	204	strcpy.....	85
relalloc.....	205	strcspn.....	86
remove.....	167	strlen.....	87
rename.....	168	strncat.....	88
repmem.....	169	strncmp.....	89
rewind.....	52	strncpy.....	90
rlsmem.....	206	strpbrk.....	91
rlsml.....	207	strpos.....	92
rstmem.....	208	strrchr.....	93
sbrk.....	209	strrpbrk.....	94
scanf.....	53	strrpos.....	95
Set_Echo.....	57	strspn.....	96
Set_Tab.....	59	stscmp.....	97
setbuf.....	170	stspfp.....	181
setjmp.....	171	tan.....	125
setmem.....	210	tanh.....	126
setnbuf.....	172	tell.....	182
setpid.....	173	time.....	183
setuid.....	174	toint.....	66
setwindow.....	58	tolower.....	67
signal.....	175	toupper.....	68
sin.....	121	ttyn.....	184
sinh.....	122	ungetc.....	69
sleep.....	176	ungetch.....	70
sprintf.....	60	unlink.....	185
sqrt.....	123	vcprintf.....	71
srand.....	124	vfprintf.....	71
sscanf.....	61	vprintf.....	71
stcarg.....	188	vsprintf.....	71
stccpy.....	74	wputs.....	72
stcd_i.....	177	write.....	186
stch_i.....	178		

License Agreement

License Agreement

This manual and the software described in it were developed and are copyrighted by Symantec Corp. (Symantec) and are licensed to you on a non-exclusive, non-transferable basis. Neither the manual nor the software may be copied in whole or in part except as follows:

- 1) You may make backup copies of the software for your use provided that they bear Symantec's copyright notice.
- 2) You have the right to include object code derived from the libraries in programs that you develop using the software and you also have the right to use, distribute and license such programs to third parties without payment of any further license fees, so long as a copyright notice sufficient to protect *your* copyright in the software in the United States or any other country is included in the graphic display of your software and on the labels affixed to the media on which your software is distributed.

You may not in any event distribute any of the source files provided or licensed as part of the software. You may use the software at any number of locations so long as there is no possibility of it being used at more than one location at a time.

Symantec's Plain Language License Statement

Symantec is concerned with how you copyright your software only in the case where you use object code of libraries which Symantec provides in source form (MacTraps does not fall into this category). These libraries may be included in your program so long as a copyright notice that will protect *your* copyright in the software is in the "About box" of your software and on the disk labels, as specified in the license agreement. You are not required to include a specific Symantec copyright notice except if your copyright does not satisfy the above requirement. This is only an explanation of the License Agreement. All terms and conditions of the License Agreement apply.

Limited Warranty on Media and Manuals

If you discover physical defects in the media on which this software is distributed or in the manuals distributed with the software, Symantec will replace the media or manuals at no cost to you provided that you return the defective materials along with a copy of your receipt to Symantec or to an authorized Symantec dealer during the 60-day period following your receipt of the software.

Limited Warranty on the Product

Symantec warrants that the software will perform substantially as described in the User's Manual. If within 60 days of receiving the software, you give written notification to Symantec

Standard Libraries Reference

of a significant, reproducible error in the software which prevents operation, and provide a written description of the possible problem along with a machine readable example, if appropriate, Symantec will either provide you with corrective or workaround instructions, a corrected copy of the software, a correction to the User's Guide and Reference Manual, or Symantec will refund your purchase price upon return of all copies of the software and documentation together with a copy of your receipt. This warranty extends only to you and shall be void if the software has been tampered with, modified, or improperly used, or if the software is used on hardware other than the Apple Macintosh™ Computer.

EXCEPT FOR THE LIMITED WARRANTY DESCRIBED ABOVE, THERE ARE NO WARRANTIES TO YOU OR ANY OTHER PERSON OR ENTITY FOR THE PRODUCT EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OR MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. ALL SUCH WARRANTIES ARE EXPRESSLY AND SPECIFICALLY DISCLAIMED. Some states do not allow the exclusion of implied warranties or limitations on how long they last, and you also may have other rights that vary from state to state. IN NO EVENT SHALL SYMANTEC BE RESPONSIBLE FOR ANY INDIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR SIMILAR DAMAGES OR LOST DATA OR PROFITS TO YOU OR ANY OTHER PERSON OR ENTITY REGARDLESS OF THE LEGAL THEORY, EVEN IF WE HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Some states do not allow the exclusion or limitation of incidental or consequential damages, so the above limitation or exclusion may not apply to you. The warranty and remedies set forth are exclusive and in lieu of all others, oral or written, express or implied.

SYMANTEC MAKES NO WARRANTY OF THE PERFORMANCE OF THE LIBRARIES WHEN USED IN YOUR SOFTWARE. YOU AGREE TO INDEMNIFY SYMANTEC FROM ALL CLAIMS BY THIRD PARTIES ARISING IN CONNECTION WITH THE USE OF YOUR SOFTWARE.

General Terms This license states the entire agreement between the parties and supercedes all other communications between the parties relating to this License, which shall be governed and construed in accordance with the laws of the State of California. You agree to bring any proceeding to enforce or construe this License or involving the performance of the software only in a federal or state court residing in the State of California. The prevailing party in any such proceedings shall be entitled to recover its attorneys' fee and litigation expenses in addition to other appropriate relief. If any provision of this License by Symantec shall be held to be unenforceable such holding shall not affect the enforceability of any other provision hereof. Waiver of any breach of this License by Symantec shall not be considered a waiver of any other or subsequent breach. the licensed software is a unique and valuable asset of Symantec and Symantec has the right to seek whatever equitable and legal redress which may be available to it for your breach of the provisions of the License.

"Lightspeed" is a trademark of Lightspeed, Inc., and is used with its permission.

1

SYMANTEC

SYMANTEC™

135 South Road
Bedford, MA 01730